ARCHITECTURAL ENHANCEMENTS FOR COLOR IMAGE AND VIDEO PROCESSING ON EMBEDDED SYSTEMS

A Dissertation
Presented to
The Academic Faculty

By

Jongmyon Kim

In Partial Fulfillment
Of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
March 2005

ARCHITECTURAL ENHANCEMENTS FOR COLOR IMAGE AND VIDEO PROCESSING ON EMBEDDED SYSTEMS

Approved by:

Dr. D. Scott Wills, Advisor

Dr. Linda M. Wills, Co-advisor

Dr. Hsien-Hsin S. Lee

Dr. David Anderson

Dr. Bonnie S. Heck

Dr. Santosh Pande

Date Approved: March 30, 2005

ACKNOWLEDGEMENTS

This dissertation could not be completed without the aid and support of countless people over the past several years. I would like to thank everyone who influenced this work.

First of all, I would like to express my deep appreciation to Dr. D. Scott Wills, my dissertation advisor, for his support, encouragement, attention to detail, and guidance. It has been an honor and a pleasure to work with him during my stay at Georgia Institute of Technology. I cannot imagine a better advisor.

I am extremely thankful to Dr. Linda M. Wills, my dissertation co-advisor, for her sincere advice and encouragement. I am very grateful to her for many valuable comments and enlightening discussions we had.

I am truly grateful to Dr. Hsien-Hsin S. Lee, Dr. David Anderson, Dr. Bonnie S. Heck, and Dr. Santosh Pande for serving on my thesis committee. Their valuable advice has improved the quality of this thesis.

I wish to extend my special thanks to Dr. Haniph A. Latchman, my former advisor. I am indebted to him for many help, support, and encouragement over the many years at the University of Florida.

I also would like to extend many thanks to all members of the PICA and EASL groups, both past and present, for their helps and friendship. Their friendship has made my graduate studies more enjoyable, and it has been a great pleasure to have worked with them; especially Krit Athikulwongse and Dr. Santithorn Bunchua for their exceptional friendship, encouragement, and research collaboration; Dr. Antonio Gentile for his expert

advice and help in developing the SIMPil simulator for color image and video processing; and all others, Jinsung, Soojung, Nidhi, Hongkyu, Peter, Brett, Cory, Sanyo, Lewis, William, Tarek, and Chris, and Mark, in no particular order.

I am forever indebted to my wife Yunjung for her all love and support. She is my life.

I dedicate this thesis to my family; my parents, Mr. Chi-Gon Kim and Mrs. Ryum-Lee Kim, my wife, my sister, Ms. Jung-Hwa Kim, and my two children, Tony and Ryan. This thesis would not have been possible without their unending love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

As emerging portable multimedia applications demand more and more tremendous computational throughput with limited energy consumption, the need for high-efficient, high-throughput embedded processing is becoming an important challenge in computer architecture. In this regard, this dissertation addresses application-, architecture-, and technology-level issues in existing processing systems to provide efficient processing of multimedia in many, or ideally all, of its form. In particular, this dissertation explores color imaging in multimedia while focusing on two architectural enhancements for memory- and performance-hungry embedded applications: (1) a pixel-truncation technique and (2) a color-aware multimedia instruction set extension (CAX) for embedded multimedia systems. The pixel-truncation technique differs from similar techniques (e.g., 4:2:2 and 4:2:0 subsampling) used in image and video compression applications (e.g., JPEG and MPEG) in that it reduces information content in individual pixel word sizes rather than in each dimension. Thus, this technique drastically reduces the bandwidth and memory required to transport and store color images without a perceivable distortion of color. At the same time, it maintains the pixel storage format of color image processing in which each pixel computation is simultaneously performed on 3-D YCbCr components, which are widely used in the image and video processing community. On the other hand, utilizing parallelism within the human perceptual YCbCr space, CAX supports parallel operations on two-packed, truncated 16-bit (6:5:5) YCbCr data on a 32-bit datapath processor, providing greater concurrency and efficiency for processing color image sequences. Thus, CAX, coupled with the pixel-truncation

technique, provides an efficient mechanism for performance- and memory-hungry embedded applications.

This dissertation presents the impact of CAX on both processing performance and cost for color imaging applications in three major processor architectures: dynamically scheduled (superscalar), statically scheduled (very long instruction word, VLIW), and embedded single instruction, multiple data (SIMD) media processors. Unlike typical multimedia extensions (e.g., MMX, VIS, and MDMX), CAX obtains substantial performance and code density improvements through direct support for color data processing rather than depending solely on generic subword parallelism. In addition, CAX's ability to reduce data format size reduces system cost. The reduction in data bandwidth also simplifies system design. Experimental results on a dynamically scheduled, 4-way issue processor indicate that CAX achieves a speedup ranging from 3× to 5.8× over the baseline performance. This is in contrast to MDMX (a representative MIPS multimedia extension), which achieves a speedup ranging from only 1.6× to 3.2× over the baseline. CAX also outperforms MDMX in energy reduction (68% to 83% reduction with CAX, but only 39% to 69% reduction with MDMX over the baseline version). Furthermore, CAX exhibits higher relative performance for low-issue rates. For example, CAX achieves an average speedup of 4.7× over the baseline 1-way issue performance, but 3× over the baseline 16-way issue performance. These results demonstrate that CAX is an ideal candidate for embedded imaging systems in which high issue rates and out-of-order execution are too expensive. Similar performance results are observed for statistically scheduled processors. CAX achieves a speedup ranging from

3.3× to 6.5×, while MDMX achieves a speedup ranging from only 1.7× to 3.6× over the baseline performance on the same statistically scheduled, 4-way issue processor.

The effectiveness of CAX is much more obvious in application-specific embedded systems (e.g., embedded SIMD arrays) that aim at providing sufficient computational power for specific applications but impose strict constraint on implementation chip area and energy consumption. Experimental results using cycle accurate simulation and technology modeling indicate that CAX outperforms MDMX in speedup (5.2× to 8.9× with CAX, but only 3× to 5× with MDMX over the baseline performance) on the same representative data parallel SIMD execution platform. CAX also outperforms MDMX in both area efficiency (a 75% increase versus a 25% increase) and energy efficiency (a 75% increase versus a 24% increase), resulting in better component utilization and sustainable battery life for given system capabilities. Furthermore, CAX improves the performance and efficiency with a mere 3% increase in system area and a 5% increase in system power, while MDMX requires a 14% increase in system area and a 16% increase in system power. Overall, CAX, coupled with the pixel-truncation technique, has the potential to meet the computational requirements and cost goals for future portable multimedia products.

CHAPTER 1

INTRODUCTION

1.1 Motivation

With the proliferation of color output and recording devices (e.g., digital cameras, scanners, and monitors) and color images on the World Wide Web (WWW), a user can easily record an image, display it on a monitor, and send it to another person over the Internet. However, the original image, the image on the monitor, and the received image through the Internet usually do not match because of faulty display or channel transmission errors. Color image processing methods offer solutions to many of the problems that occur in recording, transmitting, and creating color images. Moreover, understanding the characteristics of the color imaging application domain provides new opportunities to define an efficient architecture for embedded multimedia systems.

Early digital color image processing was often approached as an extension of monochrome image processing, in which each color channel was treated as an independent monochrome image [84]. However, this approach may not be able to extract certain crucial information conveyed by color since it fails to take into account the correlation between color channels. Clearly, color cannot be treated as just another dimension, and the relationship between color channels is much more complex because of the definition of color space and human color perception.

This dissertation explores color imaging for multimedia with respect to the following issues (see Chapter 2):

- Which color specification model is most suitable for achieving a natural extension of the operation?
- What are the advantages and disadvantages of the use of color information in multimedia applications using a vector approach in which each pixel computation is performed simultaneously on three color channels?
- Are there any efficient techniques that support 3-D vector computation?

Color imaging applications demand tremendous computational throughput. Moreover, increasing user demand for color-multimedia-over-wireless capabilities on embedded systems places additional constraints on power, size, and weight.

Application-specific integrated circuits (dedicated ASICs) can meet the needed performance and cost goals for such embedded imaging systems. However, they provide limited, if any, programmability or flexibility required by emerging imaging applications.

General-purpose microprocessors (GPPs) offer the necessary flexibility and inexpensive processing elements, and multimedia extensions to GPPs have improved the performance of multimedia applications with little added cost to the processors. Examples include Intel MMX™ [67], SSE™, and SSE-2 [70], Hewlett Packard MAX-2 for the PA-RISC architecture [53], Sun VIS for SPARC [80], MIPS MDMX [60], Alpha MVI [75], and Motorola ALTIVEC for PowerPC™ architecture [63]. These extensions exploit subword parallelism by packing several small data elements (e.g., eight-bit pixels) into a single wide register (e.g., 32-, 64-, and 128-bit) while processing these separate elements in parallel within the context of a dynamically scheduled superscalar machine. The designers of digital signal processors (DSPs), such as the Texas Instruments TMS320C64x families [82] and the Analog Devices TigerSharc processor [31], have

followed the trend. While the improvement in performance has been exciting and encouraging, their performance is limited in dealing with both color data that are not aligned on boundaries that are powers of two (e.g., visually adjacent pixels from each band are spaced three bytes apart) and storage data types that are inappropriate for computation (necessitating conversion overhead before and usually following the computation) [77]. Although the band separated format (e.g., the red data for adjacent pixels are adjacent in memory) is the most convenient for single instruction, multiple data (SIMD) processing, a significant amount of overhead for data alignment is expected prior to SIMD processing. Even if the SIMD multimedia extensions store the pixel information as a packed 32-bit word composed of an eight-bit red (R), green (G), blue (B), and unused (U) field (*band interleaved* format) in a 32-bit wide register, subword parallelism can not be exploited on the operand of the unused field. Moreover, since the RGB space does not model the perceptual attributes of human vision well, the RGB to YCbCr (a human perceptual color space that is widely used in the image and video processing community) conversion is necessary for further color image and video processing [85][36]. Although the SIMD multimedia extensions can handle the color conversion process in software, the hardware approach would be much more efficient.

This dissertation proposes a color-aware instruction set extension (CAX) as a solution to the problems inherent to packed RGB extensions by supporting two-packed 16-bit (6:5:5) YCbCr data in a 32-bit register while processing these separate color data in parallel. The YCbCr space allows coding schemes that exploit the properties of human vision by truncating some of the less important data in every color pixel and allocating fewer bits to the high-frequency chrominance components that are perceptually less

significant. Thus, the compact 16-bit color representation consisting of a six-bit luminance (Y) and two five-bit chrominance (Cb and Cr) components provides satisfactory image quality [45][46]. This pixel-truncation technique differs from similar techniques (e.g., 4:2:2 and 4:2:0 subsampling) used in image and video compression applications [85] in that it reduces information contents in individual pixel word sizes, rather than in each dimension, while inheriting the chrominance components of the luminance for the vector process. In addition, CAX offers greater concurrency with minimal hardware modification. Figure 1 shows an example of how a 32-bit ALU functional unit can be used to perform either a 32-bit baseline ALU or two 6:5:5-bit ALUs. The 32-bit ALU is divided into two six-bit ALUs and four five-bit ALUs. When the output carry (Cout) is blocked (i.e., Cin = 0), the six smaller ALUs can be performed in parallel. Chapter 3 presents the impact of CAX on both performance and energy consumption for color imaging applications on dynamically scheduled superscalar processors.

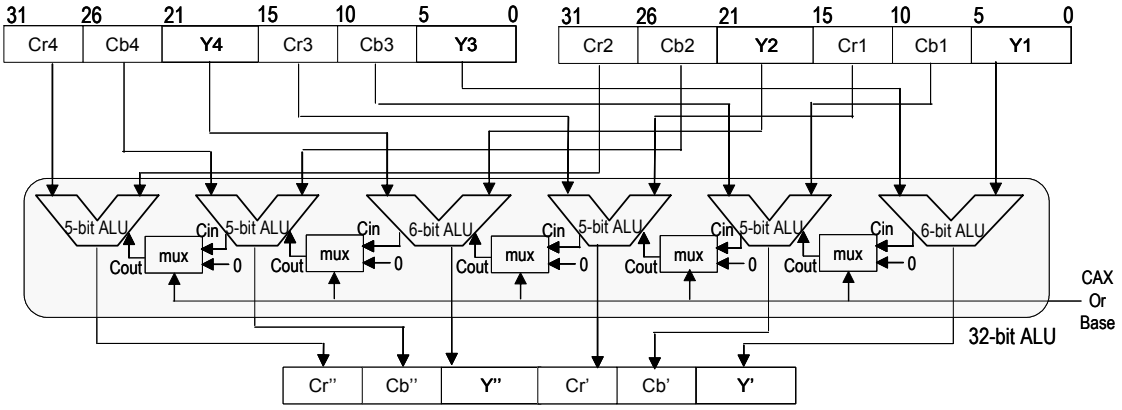


Figure 1. An example of a partitioned ALU functional unit that exploits color subword parallelism.

Despite some performance improvements through multimedia extensions, neither GPPs nor DSPs will be able to meet the higher levels of performance required by emerging multimedia applications on higher resolution images. This is because they lack the ability to exploit the full data parallelism available in these applications.

Among many computationally efficient models available for imaging applications, SIMD arrays are promising candidates for application-specific embedded systems since they replicate the datapath, data memory, and I/O to provide high processing performance with low node cost. Whereas instruction-level or thread-level processors use silicon area for large multiported register files, large caches, and deeply pipelined functional units, SIMD arrays increase the number of simple processing elements (PEs) for the same silicon area. As a result, SIMD arrays often employ thousands of PEs while possibly distributing and co-locating PEs with the data I/O to minimize storage and data communication requirements. The SIMD Pixel (SIMPil) processor [13][34][8] being developed at Georgia Tech, for example, is a low memory, monolithically integrated SIMD architecture that benefits from the efficient exploitation of data parallelism in a SIMD array, short wire lengths, and specialized microarchitecture to provide a significant improvement in energy efficiency. While 2-D SIMD arrays, including SIMPil, are well suited for many imaging tasks that require processing of pixel data with respect to either nearest-neighbor or other 2-D patterns exhibiting locality or regularity, they are less amenable to the vector processing of 3-D color channels. More specifically, since the 3-D vector computation is performed within innermost loops, its performance does not scale with larger PE arrays.

CAX efficiently eliminate this performance limitation by including parallel operations on two-packed 16-bit YCbCr data in the instruction set architecture (ISA) of the 32-bit datapath SIMD array. In addition to greater concurrency and efficiency for processing color image sequences, the ability to reduce data format size reduces system cost. The reduction in data bandwidth also simplifies system design. This dissertation presents the impact of CAX on processing performance and on both area and energy efficiency for color imaging applications in a representative SIMD array architecture with respect to the following issues (see Chapter 4):

- Existing multimedia extensions have been successful at achieving subword parallelism between loop iterations in the innermost loops of multimedia applications. What performance is possible with CAX in comparison to a representative multimedia extension, MDMX, an extension of MIPS? MDMX is chosen as a basis of comparison because it provides an effective way of dealing with reduction operations by using a wide packed accumulator that successively accumulate the results produced by operations done with multimedia vector registers. Other multimedia extensions poorly support vector processing in a 32-bit datapath processor without accumulators. To handle vector processing on a 64-bit or 128-bit datapath, they require frequent packing/unpacking of operand data, deteriorating their performance.
- For portable embedded systems, chip size and battery life are as critical as processing performance. The addition of a CAX or MDMX execution unit into a simple PE or an entire array may lead to substantial system area and

power overheads. What percentage of the system area and power overheads for the CAX and MDMX ISAs is added to the SIMD array?

- Which ISA extension between CAX and MDMX achieves higher area efficiency [Gops/(sec·mm2)] and energy efficiency [Gops/Joule]?

Another significant issue for such embedded SIMD array architectures is determining the ideal grain size that provides sufficient processing performance with the lowest cost and the longest battery life for target applications. In color imaging applications, the grain size of the PEs determines the number of vector pixels that are mapped to each PE, which is called the vector-pixel-per-processing-element (VPPE) ratio. The VPPE ratio has a significant impact on the overall area and energy efficiency of the computational array. This dissertation evaluates the effects of different VPPE ratios on performance and efficiency for a specified PE architecture and implementation technology (see Chapter 5). The impact of CAX on each VPPE configuration is also evaluated to identify optimal grain sizes that yield the most efficient PE granularity.

Overall, the research presented in this dissertation explores color imaging for multimedia in existing processing systems while focusing on two architectural techniques for memory- and performance-hungry embedded applications: (1) a pixel-truncation technique and (2) a new color-aware multimedia instruction set extension (CAX) for embedded multimedia systems.

1.2 Problem Statement and Contributions

Color image and video processing has garnered considerable interest over the past few years since color features are valuable in sensing the environment, recognizing

objects, and conveying crucial information. As a result, color imaging applications now define a significant portion of the computing market. However, the behavior of color imaging for multimedia on existing processing systems is not well understood, as discussed in the previous section. Thus, the efficient processing of color image sequences is one of the key issues in the multimedia processing application domain. Designing "color-aware" embedded systems requires a study of applications, architectures, and technologies to provide efficient processing of color multimedia in many, or ideally all, of its forms. The research presented in this dissertation addresses application-, architecture-, and technology-level issues in existing processing systems to support color image and video processing with the following approaches:

- Evaluate several color specification models to identify the most suitable model that achieves a natural extension of the operation.
- Investigate the use of color information in multimedia applications using a vector approach.
- Evaluate several color representations with varying pixel word sizes through a pixel-truncation technique to determine the most efficient representation in terms of storage requirements and color accuracy.
- Develop an efficient color-aware instruction set extension (CAX) for embedded color image and video processing.
- Introduce the CAX plus pixel-truncation technique to modern processor architectures, including dynamically scheduled, statistically scheduled, and embedded SIMD array processors.

- Model the new technique through detailed execution-driven simulators and evaluate the performance of the new approach.
- Develop a hardware implementation cost model of the new approach using architectural and technology modeling tools.
- Combine the execution performance and implementation cost of the new approach to determine overall processing performance, area efficiency, and energy efficiency.

1.2.1 Exploring Color Imaging for Multimedia

With the proliferation of color imaging devices and wireless computer networks, consumer demand for color-multimedia-over-wireless capabilities on embedded systems is growing rapidly. As a result, the requirements of color imaging applications in terms of computations, storage, and communications pose a new set of design constraints on existing systems. Thus, understanding the characteristics of color imaging for multimedia provides new opportunities to define an efficient and reconfigurable architecture for embedded multimedia systems.

In this research, several color specification models are evaluated to determine the most advantageous model that achieves the most effective results in color image processing. In addition, the use of color information in multimedia applications using a vector approach is investigated, improving the accuracy of the process and overall image quality. Furthermore, several color representations with varying pixel word sizes are evaluated to identify the most efficient representation in terms of storage requirements and color accuracy. In particular, a 16-bit (6:5:5) YCbCr representation is examined for

reduced-memory, embedded video processing. The 16-bit YCbCr representation reduces pixel word storage by 33% over the baseline 24-bit YCbCr representation. Overall image quality remains high, and color imaging applications continue to perform well using the reduced storage format.

1.2.2 Utilizing Color Subword Parallelism in Superscalar ILP Processors

Application-specific extensions of a processor provide an efficient mechanism to meet the growing performance demands of multimedia applications. In this research, a new color-aware instruction set extension (CAX) for dynamically scheduled superscalar processors is presented to improve the performance of color imaging applications. Unlike typical multimedia extensions (e.g., MMX, VIS, and MDMX), CAX obtains substantial performance and code density improvements by direct support for color data processing. Rather than depending solely on generic subword parallelism, CAX supports parallel operations on two-packed 16-bit (6:5:5) YCbCr data in a 32-bit datapath processor, providing greater concurrency and efficiency for processing color image sequences. Some of the key findings follow. CAX achieves a speedup ranging from 3× to 5.8× over the baseline performance on a dynamically scheduled, 4-way issue superscalar processor. This is contrast to MDMX (a representative MIPS multimedia extension), which achieves a speedup ranging from only 1.6× to 3.2× over the baseline. CAX also outperforms MDMX in energy reduction (68% to 83% reduction with CAX, but only 39% to 69% reduction with MDMX over the baseline version). Moreover, CAX exhibits higher relative performance for low-issue rates. For example, CAX achieves an average speedup of 4.7× over the baseline 1-way issue performance, but 3× over the baseline 16-way issue

performance. These results demonstrate that CAX is an ideal candidate for embedded multimedia systems in which high issue rates and out-of-order execution are too expensive.

1.2.3 Implementation and Evaluation of the Color-Aware Instruction Set for Low-Memory, Embedded Video Processing in Data Parallel Architectures

Future embedded imaging products must achieve greater processing performance while maintaining low cost and low energy consumption. Data parallel architectures (e.g., embedded SIMD arrays) have demonstrated the potential to meet the computational requirements and cost goals by employing thousands of inexpensive processing elements and possibly distributing and collocating PEs with the data I/O to minimize storage and data communication requirements. While 2-D SIMD arrays exploit massive data parallelism inherent in image sequences by operating the same instruction sequences simultaneously on a large number of discrete data sets, they are less amenable to the vector processing of 3-D YCbCr channels, which are widely used in image and video processing community. In particular, since the 3-D vector computation is performed within innermost loops, its performance does not scale with increasing PEs in the computational array.

CAX is presented as a solution to this performance limitation by adding parallel operations on two-packed 16-bit (6:5:5) YCbCr data to the instruction set architecture of the 32-bit datapath SIMD array. In addition to greater concurrency, the ability to reduce data format size reduces system cost. The major findings are the following:

- CAX outperforms MDMX across all the selected programs in speedup (5.2× to 8.9× with CAX, but only 3× to 5× with MDMX over the baseline performance) on the same data parallel SIMD execution platform.
- CAX also outperforms MDMX in both area efficiency (a 75% increase versus a 25% increase) and energy efficiency (a 75% increase versus a 24% increase), resulting in better component utilization and sustainable battery life.
- Furthermore, CAX improves the performance and efficiency with a mere 3% increase in the system area and a 5% increase in the system power, while MDMX requires a 14% increase in the system area and a 16% increase in the system power. These results demonstrate that CAX is a suitable candidate for application-specific embedded systems.

1.2.4 Analytically Determining Optimal Grain Sizes in Embedded SIMD Architectures

Reconfigurable silicon area usage within an integrated pixel processing array is a key issue for focal-plane SIMD imaging architectures because of limited chip resources and varying application requirements. This research explores the effects of varying the VPPE ratio (number of vector pixels mapped to each processor within a SIMD architecture) on processing performance and on both area and energy efficiency for a specified PE architecture and implementation technology. The impact of CAX on each VPPE configuration is also evaluated to identify the most efficient PE granularity. Experimental results using cycle accurate simulation and technology modeling indicate that CAX outperforms MDMX for all the configurations for full search vector quantization in terms of processing performance, area efficiency, and energy reduction.

The results also indicate that high processing performance with the lowest cost is achieved at VPPE = 16 with CAX.

1.2.5 Static versus Dynamic Scheduling

With limited amounts of memory and register sizes tailored for specific applications and low cost, early media processor designs have followed the digital signal processor design philosophy, building processors with predominantly static architectures, such as VLIW architectures. However, as media processors progress to higher frequencies and a higher degree of parallelism with the increasing number of gates made available as predicted by Moore's Law, the dynamic aspects of processing are becoming more pronounced. Architectures employing dynamic scheduling, such as superscalar architectures, may be conducive to emerging multimedia applications [32].

This research compares the performance of static versus dynamic architectures with and without CAX or MDMX for color imaging applications through a common simulation framework. Experimental results using the Simplescalar-based simulator and a retargeting tool indicate that the dynamic approach with a four-way issue achieves an average speedup of 2.7× over the static approach with a four-way issue. This is primarily because the static code schedules are poorly adapted to the run-time conditions of the processor. CAX achieves an additional speedup of 7.6×, while MDMX achieves an additional speedup of 2.7×.

1.3 Contribution Summary

The contributions made in this dissertation include the study of color imaging algorithms, architectures, and technologies to provide efficient processing of color multimedia in many, or ideally all, of its forms on embedded multimedia systems. The contributions are outlined in the five categories below.

1.3.1 Exploring Color Imaging for Multimedia

- Evaluation of several color specification models for determining the most suitable color space model that achieves a natural extension of the operation.
- Investigation of the use of color information in multimedia applications using a vector approach, improving the accuracy of the process and overall image quality.
- Evaluation of color representations (e.g., YCbCr) with varying pixel word sizes for identifying the most efficient color representation in terms of storage requirements and color accuracy.

1.3.2 Utilizing Color Subword Parallelism in Superscalar ILP Processors

- Design and definition of the CAX instruction set for dynamically scheduled processor architectures.
- Validation of CAX effectiveness in capturing the intended workload.
- Evaluation of the CAX instruction set on performance and energy consumption through detailed execution-driven simulators (e.g., Simplescalar out-of-order superscalar modeling and Wattch power modeling).

- Comparison of the execution performance and energy consumption of CAX versus MDMX (a representative MIPS multimedia extension).

1.3.3 Implementation and Evaluation of the Color-Aware Instruction Set for Low-Memory, Embedded Video Processing in Data Parallel Architectures

- Design and definition of the CAX instruction set for embedded SIMD array architectures.
- Development of a detailed execution-driven SIMD simulator that supports CAX and MDMX instruction set extensions.
- Validation of CAX effectiveness in capturing target applications.
- Evaluation of the impact of CAX on processing performance and on both area and energy efficiency with respect to color imaging applications.
- Performance, area efficiency, and energy efficiency comparisons against MDMX on the same data parallel SIMD architecture.

1.3.4 Analytically Determining Optimal Grain Sizes in Embedded SIMD Architectures

- Introduction of a vector-pixel-per-processing-element (VPPE) ratio.
- Illustration of the correlation among problem size, VPPE ratio, and PE architecture.
- Modification of the cycle-accurate SIMD simulator to support a different VPPE ratio and a different amount of local memory.
- Application mapping for different VPPE values, with and without CAX or MDMX, and simulations.

- Evaluation of the performance, area efficiency, and energy efficiency for each VPPE configuration with and without CAX or MDMX extensions.
- Identification of the most efficient PE granularity that delivers sufficient processing performance with the lowest cost under technology constraints.

1.3.5 Dynamic versus Static Scheduling

- Development of a common framework infrastructure that consists of the Simplescalar-based simulator and a retargeting tool.
- Performance evaluation and comparison of dynamic versus static architectures, both with and without CAX or MDMX extensions.

1.4 Overview of Content

The rest of this dissertation is organized as follows. Chapter 2 explores color imaging for multimedia. Several color space representations are first evaluated to identify the most advantageous color space that achieves the most effective results in color image processing. The use of color information in multimedia applications is then investigated along with three important imaging applications (e.g., the vector median filter, color edge detection, and motion estimation). Several color representations with varying pixel word sizes are also evaluated to determine the most efficient representation in terms of storage requirements and color accuracy.

Chapter 3 presents a color-aware instruction set extension (CAX) for dynamically scheduled superscalar processors to support the vector processing of color image sequences. Existing multimedia extensions are first presented along with research efforts

using the multimedia extensions. An overview of CAX is then introduced along with pictorial examples. Then the effectiveness of CAX is evaluated with respect to a set of color imaging applications. CAX is also compared with MDMX in terms of processing performance and energy consumption.

Chapter 4 presents the implementation and evaluation of the CAX instruction set for low-memory, embedded video processing in data parallel architectures. Research dealing with harnessing data-level parallelism (DLP) inherent in color imaging applications is first presented. The modeled architectures and a methodology infrastructure are then illustrated for the evaluation of CAX. Then the impact of CAX on processing performance and on both area and energy efficiency on a representative SIMD array is evaluated using cycle accurate simulation and technology modeling. Processing performance, area efficiency, and energy efficiency comparisons against MDMX are also provided.

Chapter 5 presents an analytical study for determining optimal grain sizes for a specified PE architecture and implementation technology. A summary of related research regarding the grain size design is first presented. The correlation among problem size, VPPE ratio, and PE architecture is then illustrated to quantify the amount of image data directly mapped to each processing element. Then the effects of varying VPPE ratio on processing performance and efficiency are evaluated. The impact of CAX is also evaluated on each VPPE configuration to identify the most efficient PE granularity that provides sufficient processing performance with the lowest cost and the longest battery life.

Chapter 6 presents a summary of this dissertation along with a list of contributions and results. A list of future research is also provided.

Appendix A presents a performance comparison of static versus dynamic architectures with and without CAX or MDMX. A methodology infrastructure is first introduced that allows both dynamically and statically scheduled simulations. The execution performance of the dynamic approach is then compared with that of the static approach. The impact of CAX on performance for both statically and dynamically scheduled programs is also provided.

Appendix B presents an in-depth description of CAX along with programming models.

CHAPTER 2

EXPLORING COLOR IMAGING FOR MULTIMEDIA

2.1 Introduction

Color image and video processing has garnered considerable interest over the past few years since color features are valuable in sensing the environment, recognizing objects, and conveying crucial information [69]. As a result, color imaging applications now define a significant portion of the computing market. Thus, understanding the characteristics of the color imaging application domain provides new opportunities to define an efficient architecture for embedded multimedia systems.

Early digital color image processing was often approached as an extension of monochrome image processing, in which each color channel was treated as an independent monochrome image [84]. However, this approach may not be able to extract certain crucial information conveyed by color because it fails to account for the correlation between color channels. Clearly, color cannot be treated as just another dimension, and the relationship between color components is much more complex due to the definition of color spaces and human perception of color.

This chapter first evaluates several color specification models with varying subsampling factors to determine the most suitable color space that consistently reduces pixel information while providing satisfactory image quality. Experimental results indicate that the luminance-chrominance (YCbCr) space performs the best out of several well-known color models (e.g., RGB, YCbCr, HSV, and L*a*b*) for all test images because the human eye is less sensitive to high frequencies in chrominance. Another

implication is that the luminance (Y) component of an image can be processed independently from its chrominance components. As a result, separate channel processing and luminance-only-processing are widely used in color imaging applications, yielding usable results [37][47][19][50]. However, both of these approaches fail to extract certain crucial information conveyed by color, reducing the accuracy of the process. It is clear that a proper vector approach to color manipulation is potentially much more beneficial.

This chapter investigates the use of color information in multimedia applications using the vector approach, improving the accuracy of the process and overall image quality. However, the major disadvantage of the vector approach is adding computational complexity to the process since the relationship between color channels is much more complex. The computational burden is further exacerbated by higher imaging resolutions, which also demand larger storage requirements. Since this storage (buffers, registers, and caches) consumes a large percentage of silicon area, the ability to reduce data format size can provide a reduction in system cost. The reduction in data bandwidth can also simplify system design.

This chapter evaluates several color representations using a pixel-truncation technique to identify the most efficient representation in terms of storage requirements and color accuracy. The pixel-truncation differs from similar techniques (e.g., 4:2:2 and 4:2:0 subsampling) [85] in that it reduces information content in individual pixel word sizes rather than in each dimension while inheriting the chrominance components of the luminance. Hence, this technique drastically reduces the bandwidth and memory required to transport and store color images while maintaining the data structure of vector processing. In particular, a 16-bit (6:5:5) YCbCr representation is examined for reduced-

memory, embedded video processing. The 16-bit YCbCr representation reduces pixel word storage by 33% over the 24-bit YCbCr representation while maintaining acceptable performance with respect to peak signal-to-noise ratio (PSNR). Moreover, this reduced pixel format is useful for an efficient color-aware instruction set (CAX) design. CAX supports parallel operations on two-packed 16-bit YCbCr data in a 32-bit datapath processor, providing greater concurrency and efficiency for processing color image sequences.

The rest of this chapter is organized as follows. Section 2.2 presents color specification models and their applications. Section 2.3 evaluates these color space models with varying subsampling factors to determine the most suitable color space that consistently reduces pixel information without perceivable distortion in color. Section 2.4 investigates the use of color information in multimedia applications using a vector approach. Section 2.5 evaluates several color representations with varying pixel word sizes to identify the most efficient representation in terms of storage requirements and color accuracy. Section 2.6 concludes this chapter.

2.2 Color Specification Models and Applications

Most color space models in use today are oriented toward either hardware or applications in which color manipulation is a goal. The models can be classified into two types: additive and subtractive. Additive color models produce color through the combination of the three primary colors: red (R), green (G), and blue (B). Examples that use additive color models include cathode-ray tube (CRT) and projection video systems. Unlike additive color models, subtractive color models create new color by subtracting

unwanted spectral components from white. Thus, subtractive environments are reflective in nature (i.e., color is displayed by reflecting light from an external source). Examples that use subtractive color models include color printers and color slides. Table 1 summarizes the most popular color space models and some of their applications. More information is available in [69].

Table 1. Color space models and their applications.

Color Space Models		Applications
Hardware-oriented	♦non-uniform spaces RGB, YIQ, YUV, YCbCr	storage, color TV broadcasting, processing, analysis coding
	♦uniform spaces L*a*b*, L*u*v*	color difference analysis, color management systems
Application-oriented	HIS, HSV, LHS	color image manipulations, computer graphics

The RGB Color Space

The most commonly used hardware-oriented color space is the RGB representation, which is widely used in color monitors and color video cameras. RGB, an additive color space, is created by mapping the three primary colors onto a 3-D Cartesian coordinate system. Color imaging files using the RGB space represent each pixel as a color triplet that consists of three numerical values in the form (R,G,B). For a 24-bit color, the triplet (0,0,0) represents black, while (255,255,255) represents white. While the RGB space is widely used to represent the image, it does not model the human perception of color well. Applying image processing techniques in the RGB space often produces color distortion and artifacts [87].

The YCbCr Color Space

Another commonly used hardware-oriented color space is the YCbCr representation, which is widely used in commercial color TV broadcasting and video systems. In the YCbCr space, Y corresponds to the luminance, and Cb and Cr are chrominance components that are used to represent hue and saturation. The YCbCr space is defined as a linear transformation applied to RGB values. Since the YCbCr space allows coding schemes to exploit the properties of human vision by allocating significantly less bandwidth to high frequency chrominance information that is perceptually less significant, the chrominance information can be subsampled without introducing a perceivable distortion of color. Another implication is that the luminance (Y) component of an image can be processed independently from its chrominance components (Cb and Cr). Other similar color spaces include YUV and YIQ in which U, V, I, and Q are chromatic components.

The L*a*b* Color Space

The L*a*b* space is very useful in applications in which precise quantification of perceptual distance between two colors is necessary [69]. The three parameters represent the perceived lightness (L*), its position between red and green (a*) and its position between yellow and blue (b*). The L*a*b* space is the uniform color space standardized by CIE, and it is designed to map perceived color differences into a Euclidean color distance metric [61].

The HSI Color Space

The commonly used application-oriented color space is the hue, saturation, and intensity (HSI) representation, which is useful for the user specification and recognition

of color. As in the YCbCr space, the intensity (I) component in the HSI space is decoupled from the chrominance information represented as hue (H) and saturation (S). Moreover, the H and S components are intimately related to the way in which human beings perceive color [36]. Thus, the HSI space is an ideal color space model for image processing applications in which the hue and saturation components are of important rather than the overall color perception. The hue, saturation, and value (HSV) space and the hue, saturation, and luminosity (HSL) space are similar to HSI in that they produce color by altering hue and saturation with the intensity.

2.3 Evaluating Color Specification Models

Color specification models are of paramount importance in applications in which efficient manipulation and communication of image frames are required [69]. This section evaluates several color space models with varying subsampling factors to identify the most suitable color space that consistently reduces pixel information while providing satisfactory image quality. Several empirical metrics and subjective comparisons are considered.

2.3.1 An Experimental Comparison of Color Space Models

A color imaging simulator, called "CISim", has been developed to evaluate color space models with varying subsampling factors using MATLAB [58]. CISim, shown in Figure 2, allows the displaying input and output images, the calculating the mean square error (MSE) and peak signal-to-noise ratio (PSNR) values, the converting color spaces, and the subsampling of any of the three components of an image. Horizontal and vertical

subsampling can reduce the resolution by averaging squares of length two, four, eight, or sixteen pixels. CISim also allows truncating pixel word sizes and processing of the three different versions (e.g., vector processing, separate channel processing, and luminance only processing) of color imaging applications, which are presented in Sections 2.5 and 2.4, respectively.

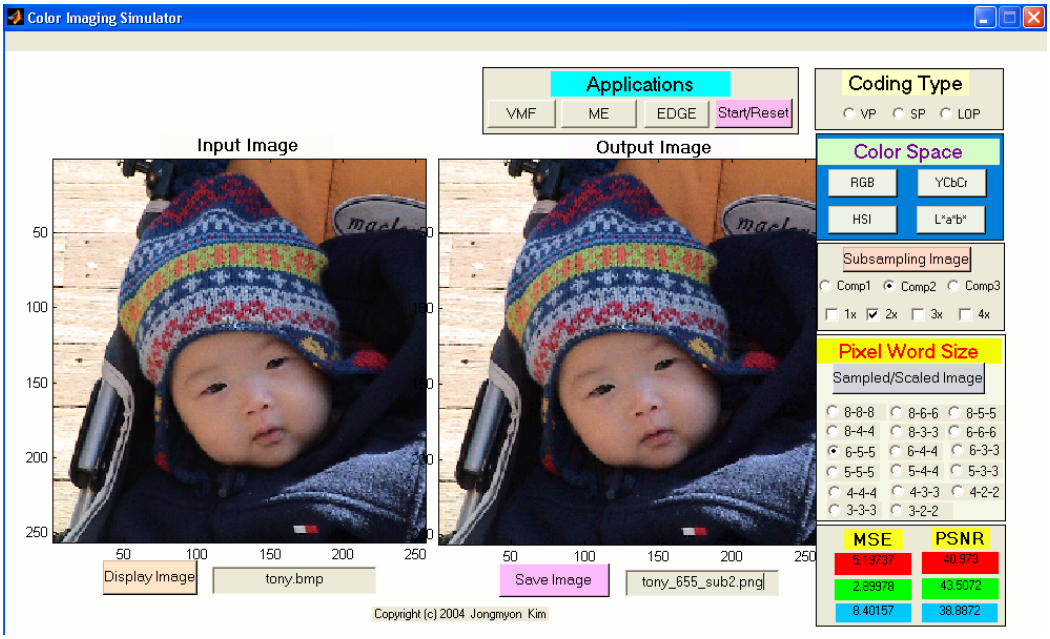


Figure 2. A screenshot of the color imaging simulator.

2.3.1.1 Experimental Results

In the first experiment, subsampling by a factor of four both vertically and horizontally is performed on each channel for each color space.

The RGB Color Space

The effect of subsampling is noticeable in each of the images, shown in Figure 3. As expected, the image subsampled in the green (G) channel is more distorted than that subsampled in the red (R) or blue (B) because the human eye is more sensitive to high frequencies in the G channel.

(a) (b) (c)

Figure 3. Subsampled images [21] with a subsampling factor of four in each direction for each component: (a) red, (b) green, and (c) blue.

The YCbCr Color Space

Distortion is hardly noticeable when the chrominance channels (Cb and Cr) are subsampled, shown in Figures 4(b) and (c). This is because the human eye is less sensitive to chrominance components. However, the image is noticeably distorted when subsampling is performed on the luminance (Y) component, shown in Figure 4(a).

(a) (b) (c)

Figure 4. Subsampled images [21] with a subsampling factor of four in each direction for each component: (a) Y, (b) Cb, and (c) Cr.

The L*a*b* Color Space

The results are similar to the YCbCr results. The image is affected by the subsampling process in the luminance (L*). However, the effect of subsampling is hardly

noticeable when the chrominance components (a* and b*) are subsampled, shown in Figure 5.

(a) (b) (c)

Figure 5. Subsampled images [21] with a subsampling factor of four in each direction for each component: (a) L*, (b) a*, and (c) b*.

The HSI Color Space

The effect of subsampling is significantly noticeable when the hue (H) is subsampled, shown in Figure 6(a). This is because perceptually different colors lie close to one another in the Euclidean plane. However, the image is slightly distorted when the intensity (I) component is subsampled, and a little distortion is observed when subsampling is performed on the saturation (S) component, shown in Figures 6(c) and (b), respectively.

(a) (b) (c)

Figure 6. Subsampled images [21] with a subsampling factor of four in each direction for each component: (a) H, (b) S, and (c) I.

In the next experiment, two components (RB in RGB, SI in HSI, CbCr in YCbCr, and a*b* in L*a*b*) in each color space are simultaneously subsampled by a factor of four both vertically and horizontally. In other words, the subsampling components are formed by blocks of 4 × 4 pixels that have the same value. Figure 7 shows subsampling results for each color space. From the previous experiment, the HSI space is ruled out because of significant distortion in color when the hue (H) channel is subsampled. Even if the S and I components are subsampled, the effect of subsampling is even more noticeable, shown in Figure 7(c). Significant distortion is also observed when the R and B components in the RGB space are subsampled. This is because each R, G, and B component is highly correlated with each other. On the other hand, the image is only slightly distorted when subsampling is performed in the chrominance channels (a*, b*, Cb, and Cr) of the L*a*b* or YCbCr spaces. However, as the subsampling coefficient increases, color at the edges is distorted for the L*a*b* space, shown in Figure 7(d). The same results have been observed for other test images, which are available at [21].

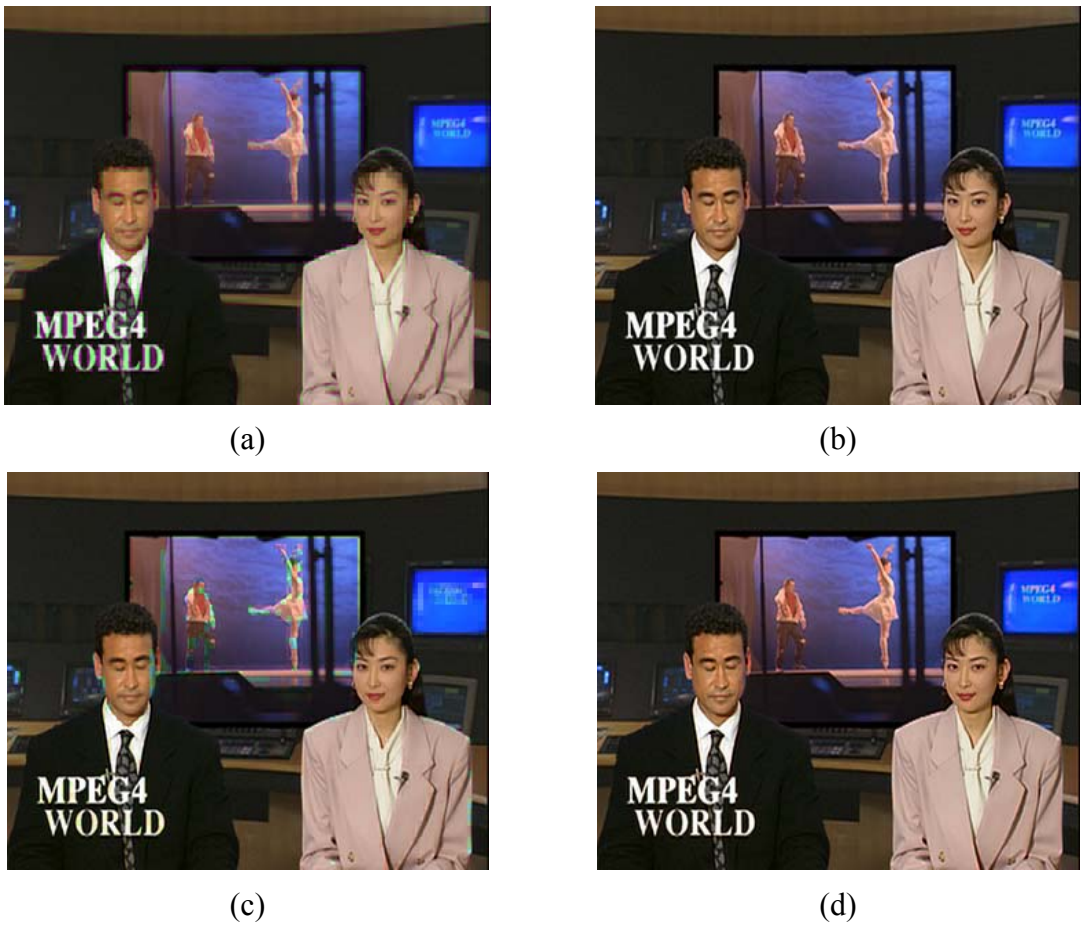


Figure 7. Subsampled images [21] with a subsampling factor of four for two components simultaneously in each color space: (a) RGB, (b) YCbCr, (c) HSI, and (d) L*a*b*.

2.3.1.2 Summary

Four commonly used color space models with varying subsampling factors have been evaluated to determine the most efficient color space that consistently reduces pixel information without perceivable color distortion. Although the RGB space is widely employed in many consumer products, it does not model the human perception of color well. On the other hand, the YCbCr space performs the best for all test images since the human eye is less sensitive to high frequencies in chrominance.

2.4 Investigating the use of Color Information in Multimedia Applications

In multichannel pixel coding, standard color images represent vector-valued image signals in which each pixel can be considered to be a vector of three components (e.g., RGB). However, as illustrated in the previous section, the RGB color space is ill-suited for the human perception of color. As a result, applying image processing techniques in the RGB space often produce color distortion and artifacts [87]. In addition, each R, G, and B component is highly correlated and thus not well-suited for independent coding. To overcome these problems, the image and video processing community widely uses the YCbCr color space, a human perceptual color space. Since the human eye is less sensitive to high frequencies in chrominance, chrominance components (Cb and Cr) can be subsampled while providing satisfactory image quality. Moreover, the YCbCr space allows luminance processing independent of chrominance channels. Because of these properties, the processing of color images can proceed by manipulating the luminance only component, shown in Figure 8(a), or each color component separately, shown in Figure 8(b). In general, these approaches provide sufficient information for the imaging process [37][47][19][50]. However, both of these approaches fail to extract crucial information conveyed by color because they do not account for the correlation between color channels, reducing the accuracy of the process. It is clear that a proper vector approach to color manipulation is potentially much more beneficial, shown in Figure 8(c). The rest of this section presents the effectiveness of the vector approach with three important applications: (1) the vector median filter, (2) color edge detection, and (3) motion estimation.

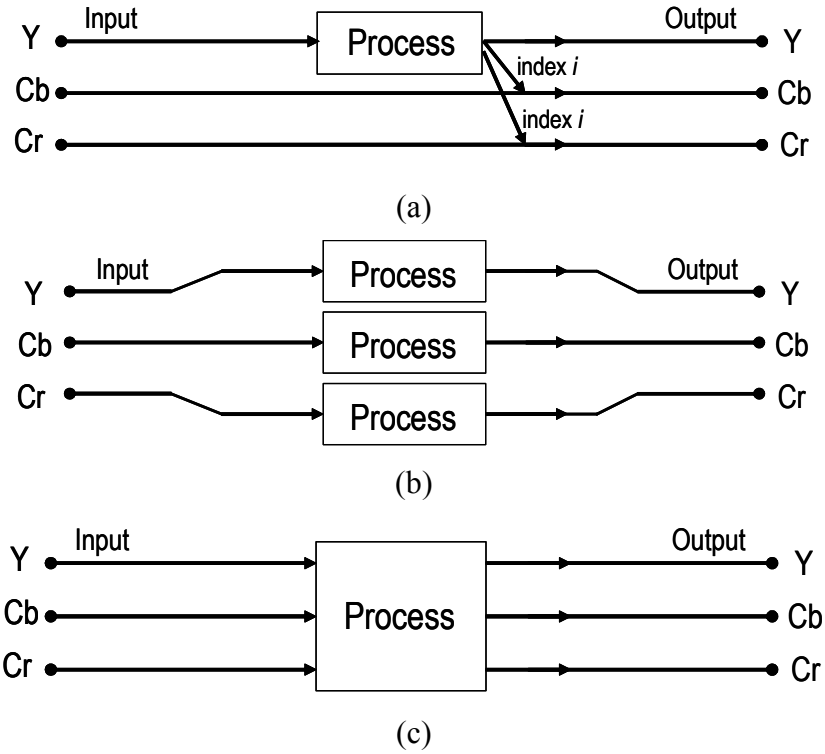


Figure 8. Three different coding schemes for color channels: (a) luminance only processing, (b) separate processing of each channel, and (c) vector processing.

2.4.1 The Vector Median Filter in the YCbCr Color Space

Impulse noise can corrupt color images due to faulty sensors or channel transmission errors. Noise reduction is an important step in color image and video processing. The most common way to filter out noise from color images is nonlinear median processing that is based on the ordering of vectors in a predefined sliding window (e.g., the well-known vector median filter proposed by Astola *et al*. [1]). The vector median filter (VMF), which is particularly effective at suppressing impulse noise in color image sequences, performed in the RGB space, does not correspond to the perceptual attributes of human vision. Therefore, this research implements the VMF on the YCbCr space to compare the YCbCr-based VMF with the luminance only median filter (LOMF) and the scalar median filter (SMF). The YCbCr-based VMF is defined as follows.

Consider a window W that is represented as a set of N color vectors $\boldsymbol{C} = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_N\}$, where each vector $\boldsymbol{p}_i = (Y_i, Cb_i, Cr_i)$, $i \in N$. This VMF computes the median pixel $\boldsymbol{p}_{VM}$ in the window, defined as

$$\boldsymbol{p}_{VM} \in \{\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_N\}, \tag{1}$$

and for all $j = 1, \ldots, N$,

$$\sum_{i=1}^{N} \| \boldsymbol{p}_{VM} - \boldsymbol{p}_i \|_1 \leq \sum_{i=1}^{N} \| \boldsymbol{p}_j - \boldsymbol{p}_i \|_1 , \tag{2}$$

where $\|\cdot\|_1$ denotes the L-1 norm.

The MATLAB tool is used to evaluate the effectiveness of the YCbCr-based VMF over the SMF and LOMF at suppressing impulse noise in color images. In the experiment, a test color frame of the *News* sequence of three-band CIF resolution (352 × 288 pixels) is corrupted by impulse noise ranging from 2% to 20% with the step size of 2% for each R, G, and B channel. The filtering results are evaluated by commonly used metrics such as the mean absolute error (MAE), the mean square error (MSE), and the normalized color distance (NCD) [69], which reflect signal preservation, noise suppression, and color chromaticity preservation, respectively. Mathematically, the MAE, the MSE, and the NCD are given by

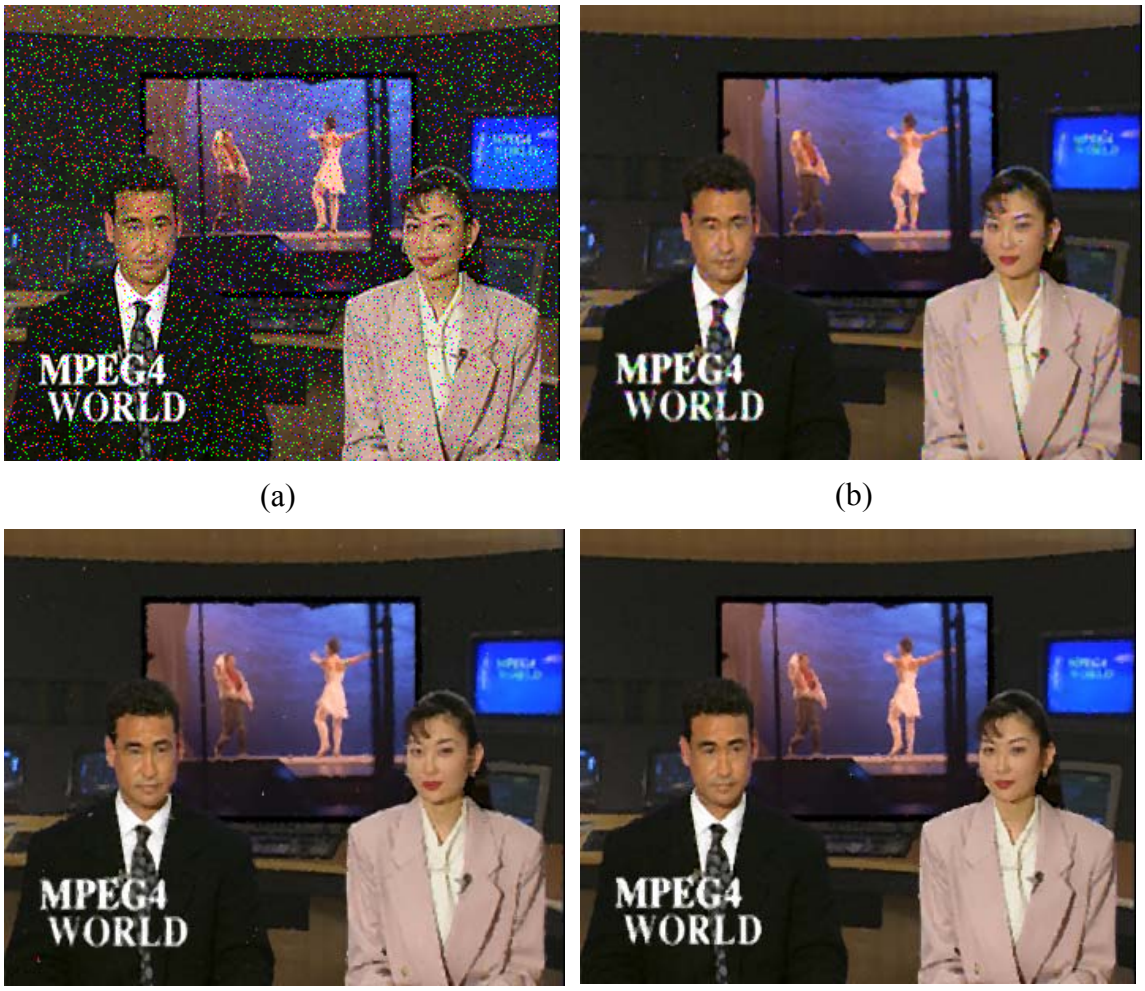
$$MAE = \frac{1}{N} \sum_{i=1}^{N} \| \boldsymbol{o}_i - \boldsymbol{x}_i \|_1 , \tag{3}$$

$$MSE = \frac{1}{N} \| \boldsymbol{o}_i - \boldsymbol{x}_i \|_2 , \text{ and} \tag{4}$$

$$NCD = \frac{\frac{1}{N} \sum_{i=1}^{N} \sqrt{(Y_i^o - Y_i^x)^2 + (Cb_i^o - Cb_i^x)^2 + (Cr_i^o - Cr_i^x)^2}}{\frac{1}{N} \sum_{i=1}^{N} \sqrt{(Y_i^o)^2 + (Cb_i^o)^2 + (Cr_i^o)^2}} , \tag{5}$$

where $\mathbf{o}_i$ is the original image pixel, $\mathbf{x}_i$ is the filtered image pixel, and $Y_i^o Cb_i^o Cr_i^o$ and $Y_i^x Cb_i^x Cr_i^x$ are values of the luminance and two chrominance components of the original image sample $\mathbf{o}_i$, and the filtered image sample $\mathbf{x}_i$, respectively.

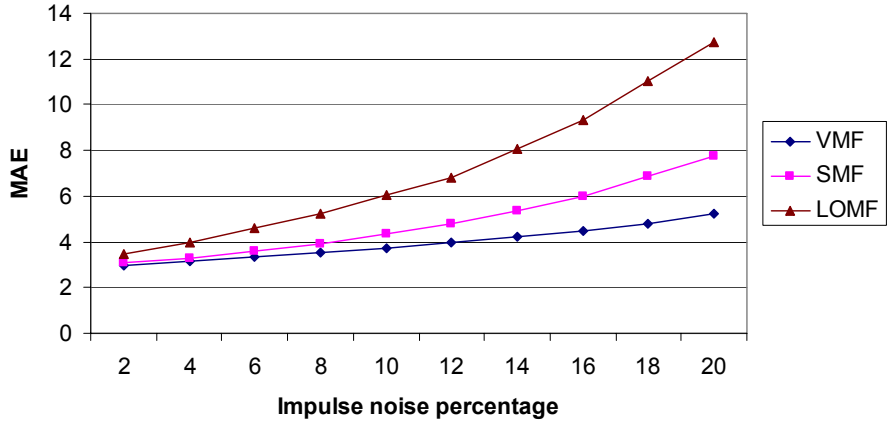
The experimental results indicate that the LOMF performs well when a small amount of impulse noise is added to an image. However, as the noise ratio increases, some noise remains at the edges, shown in Figure 9(b). Unlike the LOMF, the SMF is performed on the three color channels independently while combining the three resultant images, providing better performance at attenuating impulse noise, shown Figure 9(c). However, the SMF internally generates new vector pixels caused by the composition of reordered channel samples. Because of this, the SMF increases the MAE, MSE, and NCD values, shown in Figure 10. On the other hand, the YCbCr-based VMF takes into account the correlation between color channels, outperforming either of these approaches in the MAE, MSE, and NCD metrics, shown in Figure 10. These results demonstrate that the vector approach is necessary to provide reliable YCbCr signals for further color image and video processing.



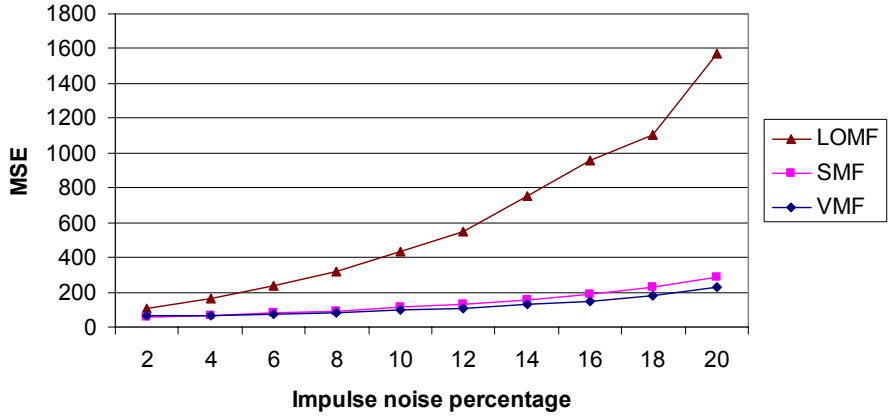
(a) (b)

(c) (d)

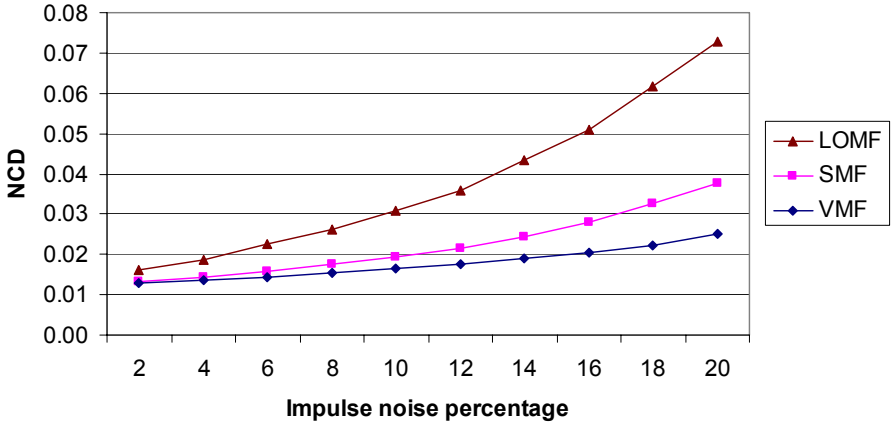
Figure 9. A corrupted image with recovered output images using relevant filters (available in color at [21]): (a) 1st frame of *News* corrupted by 8% impulse noise, (b) the luminance only median filter, (c) the scalar median filter, and (c) the YCbCr-based VMF.



(a)



(b)



(c)

Figure 10. Objective criteria in dependence on impulse noise percentage: (a) MAE, (b) MSE, and (c) NCD.

2.4.2 Color Edge Detection using both Luminance and Chrominance Components

Edge detection is a fundamental task in image processing. Many imaging applications, such as segmentation, registration, and identification of objects in a scene, depend on the accuracy of edge detection. An edge corresponds to object boundaries or changes in the physical properties such as illumination or reflectance in a monochrome image [36]. Monochrome edge detection, however, may not correspond to the set of edges existing in a color image when neighboring objects have different hues but equal intensities. The additional boundary information provided by color is crucial for applications such as object recognition and image segmentation.

For color edge detection purposes, several different approaches have been tested in [47]. The most straightforward approach is to apply monochrome edge detection to the three color channels independently. The edge results of the three channels are then combined by using a certain logical operator (e.g., fused by means of a logical *or* operator) to obtain more complete edge information. Consider, for example, the Sobel operator. The Sobel operator is implemented by convolving a pixel and its eight neighbors with two 3 × 3 convolution filters, defined as

$$M_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \quad \text{and} \quad M_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}. \tag{6}$$

The two filters are applied to each color channel independently to highlight horizontal and vertical edges, and the three resulting edge images are combined by using a logical *or* operator. Because of this, the Sobel operator provides more edge information when compared to the luminance only Sobel operator, shown in Figure 11(c). However,

this approach fails to account for the correlation between color channels, resulting in the loss of crucial information provided by color (e.g., edges that have the same strength but have opposite color components).

Unlike the scalar Sobel operator, a vector Sobel operator using the two filters shown in (7) produces vectors corresponding to the local average colors by using the Euclidean norm, shown in (8).

$$M_V = \frac{1}{4}\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \quad \text{and} \quad M_H = \frac{1}{4}\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}. \tag{7}$$

$$\text{VSobel}(x_0, y_0) = \sqrt{\| \Delta \boldsymbol{V}(x_0, y_0) \|^2 + \| \Delta \boldsymbol{H}(x_0, y_0) \|^2} \tag{8}$$

where $\|\cdot\|$ denotes the Euclidean norm, and the scalars $\|\Delta \boldsymbol{V}(x_0, y_0)\|$ and $\|\Delta \boldsymbol{H}(x_0, y_0)\|$ provide the variation rate at pixel location (x_0, y_0) in orthogonal directions (i.e., the amounts of color contrast that can be obtained in the vertical and horizontal directions).

The local average colors at pixel location (x_0, y_0) are calculated as follows:

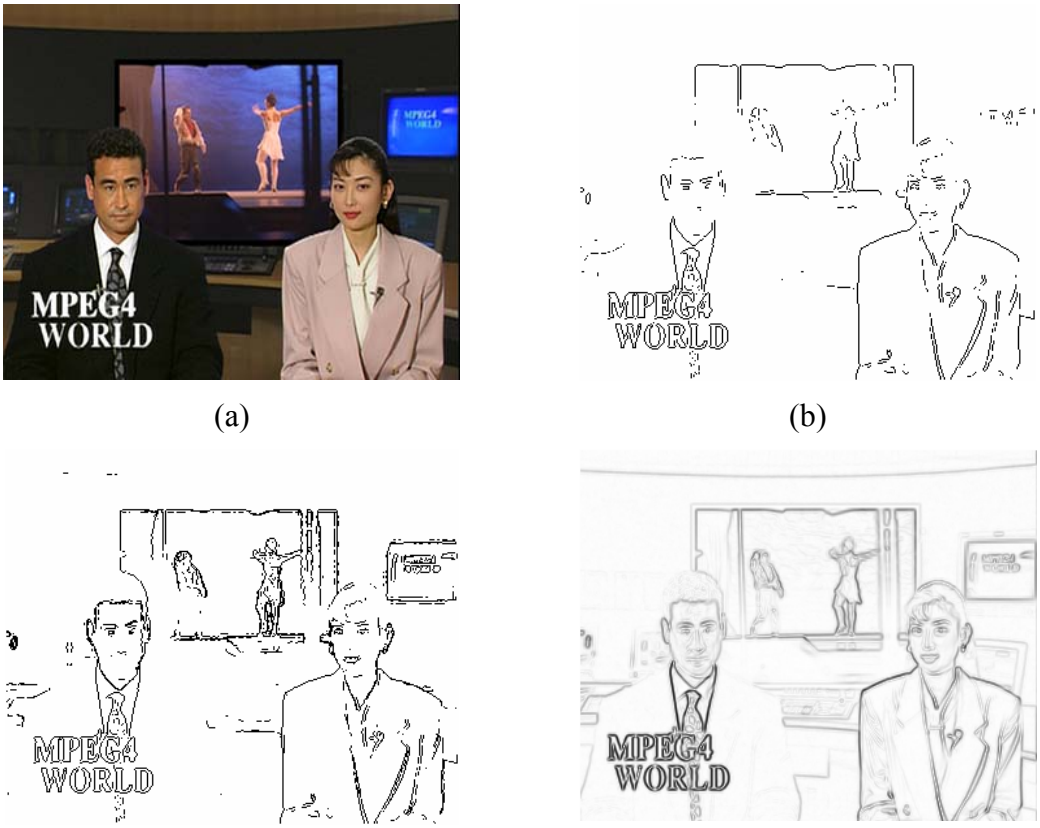
$$\Delta \boldsymbol{V}(x_0, y_0) = \sum_{y=-1}^{1} \sum_{x=-1}^{1} c(x_0 + x - 2, y_0 + y - 2) M_V(x, y), \tag{9}$$

$$\Delta \boldsymbol{H}(x_0, y_0) = \sum_{y=1}^{3} \sum_{x=1}^{3} c(x_0 + x - 2, y_0 + y - 2) M_H(x, y), \tag{10}$$

where $\mathbf{c}(x_0, y_0)$ denotes the YCbCr color vector (Y, Cb, Cr) at the image location (x_0, y_0), $\Delta \boldsymbol{V}(x_0, y_0)$ represents vectors corresponding to the vertical average colors, and $\Delta \boldsymbol{H}(x_0, y_0)$ represents vectors corresponding to the horizontal average colors.

If the local changes are combined by simply adding the Y, Cb, and Cr components of $\Delta \boldsymbol{V}$ and $\Delta \boldsymbol{H}$, this may lead to a mutual canceling out effect (e.g., when contrast is in phase opposition in different channels).

Figure 11 shows a performance comparison of the three different 3×3 edge detection algorithms: (1) the luminance only Sobel operator (LSobel), (2) the scalar Sobel operator (SSobel), and (3) the vector Sobel operator (VSobel). The qualitative results indicate that the VSobel operator provides more accurate edge information than the LSobel and SSobel operators, shown in Figure 11. Other various approaches proposed consider the problem of color edge detection in vector space [69].



(a) (b)

(c) (d)

Figure 11. Obtained output images using edge detection techniques: (a) 1st frame of *News*, (b) the luminance only Sobel operator, (c) a scalar Sobel operator, and (d) a vector Sobel operator.

2.4.3 Simultaneous Motion Estimation of All Color Components

The most important step in estimating the quality of color video frames is that of motion estimation, one of the most computationally intensive tasks in today's compression standards [19]. Motion estimation is typically done by block matching that subdivides the current frame into small reference blocks and then finds the best match for each block among available blocks in the previous frame. The standard full search block matching algorithm (FSBMA) for motion estimation uses only the luminance or intensity information of video signals to reduce the computational complexity of the process. In general, the use of only the luminance component in estimating the motion field of a color sequence provides sufficient information for the operations [50]. However, for color video frames that have low luminance or detailed color information, accurate motion estimation requires chrominance components, in which this chapter investigates. In particular, a vector approach, called the *full search vector BMA* (FSVBMA), uses both luminance and chrominance components while arriving at one motion vector for all components. The matching criterion of the VFSBMA is defined as

$$\begin{aligned} MAD(m,n) = & \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} | \, y(i+m, j+n) - x(i,j) \, |_{Y} \\ & + \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} | \, y(i+m, j+n) - x(i,j) \, |_{Cb} \\ & + \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} | \, y(i+m, j+n) - x(i,j) \, |_{Cr} \end{aligned} \quad (11)$$

$$-p \le m, n \le p \,, \quad (12)$$

$$\boldsymbol{v} = \arg\min_{-p \le m, n \le p} MAD(m,n), \quad (13)$$

where *x(i,j)* is the reference block of size $M \times N$ pixels at coordinates *(i,j)*, *y(i+m,j+n)* is the candidate block within a search area in the previous frame, *(m,n)* represents the candidate displacement vector, and $\boldsymbol{v}$ is the motion vector.

To evaluate the effectiveness of the FSVBMA, two motion estimations (e.g., the standard FSBMA and the FSVBMA) are implemented and simulated using MATLAB for three well-known color videos (*Foreman*, *News*, and *Football*). Each video contains forty frames of three-band CIF resolution (352×288) pixels. In the experiment, a macroblock of 16×16 pixels and a search range of ±8 are used. The search area in the previous frame is explored for each reference block in the current frame to find the closest matching block to a selected error criterion. Figures 12, 13, and 14 present the sum of absolute errors for the FSVBMA, normalized to the standard FSBMA for motion estimation. The results indicate that the FSVBMA outperforms the luminance only motion estimation in the sum of absolute errors for all test videos, improving the accuracy of the process and overall video quality.

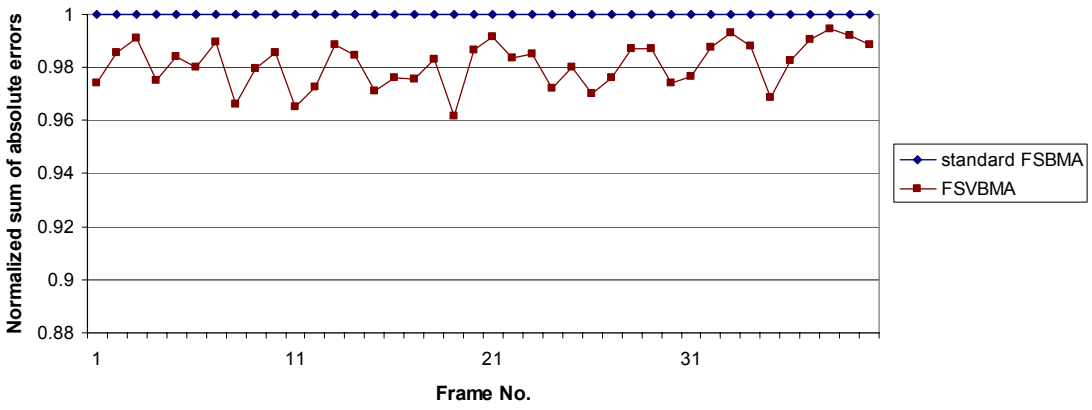


Figure 12. Sum of absolute errors of the FSVBMA for the *Foreman* video, normalized to the standard FSBMA.

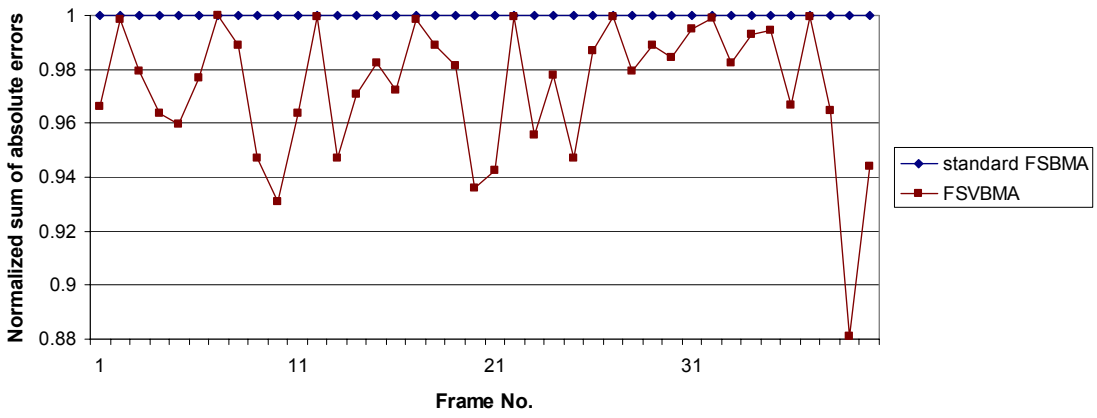


Figure 13. Sum of absolute errors of the FSVBMA for the *News* video, normalized to the standard FSBMA.

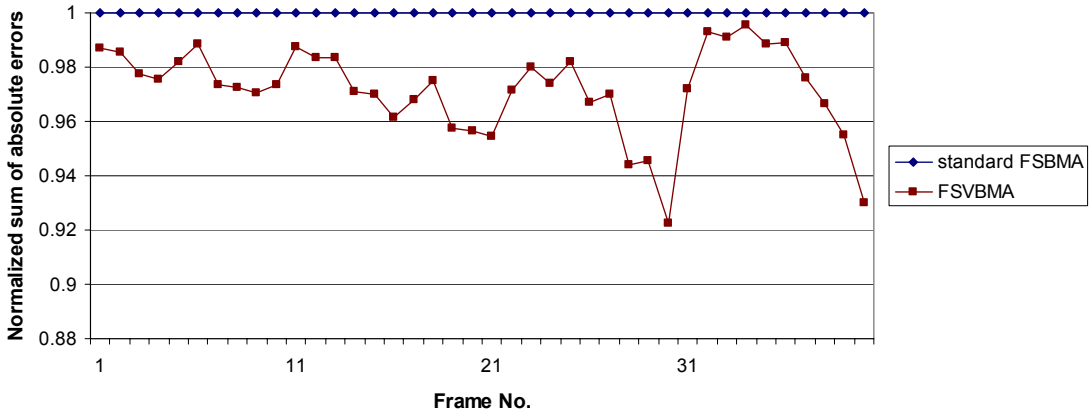


Figure 14. Sum of absolute errors of the FSVBMA for the *Football* video, normalized to the standard FSBMA.

Overall, vector processing outperforms either separate channel processing or luminance only processing in terms of the accuracy of the process and overall image quality. However, the main disadvantage of the vector approach is the addition of computational complexity to the process since the relationship between color components is much more complex. The computational burden is further exacerbated by higher imaging resolutions. Higher resolution images also require larger storage. The next two sections address these problems by introducing two architectural enhancements for

memory- and performance-hungry embedded applications: (1) a pixel-truncation technique and (2) a color-aware instruction set for embedded multimedia systems.

2.5 Determining an Efficient Color Representation using a Pixel-Truncation Technique for Low-Memory, Embedded Video Processing

Multimedia-on-a-chip solutions offer greater integration and processor-memory bandwidth. However, the trend towards higher resolution images results in higher data rates and increasing storage requirements of processors. Since this storage (buffer, registers, and caches) consumes a large percentage of silicon area, the ability to reduce data format size can provide a reduction in system cost. The reduction in data bandwidth can also simplify system design and packaging. This section evaluates several YCbCr representations with varying pixel word sizes through a pixel-truncation technique to identify the most efficient representation in terms of storage requirements and color accuracy. The pixel-truncation technique differs from similar techniques (e.g., 4:2:2 and 4:2:0 subsampling) used in image and video compression applications in that it reduces information content in individual pixel word sizes rather than in each dimension while inheriting the chrominance components of the luminance for the vector process. Several empirical metrics and subjective comparisons are considered.

2.5.1 Analysis of the YCbCr Representations with varying Pixel Word Sizes

Figures 15 and 16 show the MSE and PSNR values [3], respectively, from seven original images for various pixel word sizes of the YCbCr data. The results indicate that for those having greater than or equal to five bits in all three channels, the MSE values

are quite small, and the PSNR values are reasonably high (more than 33 dB for each R, G, and B component).

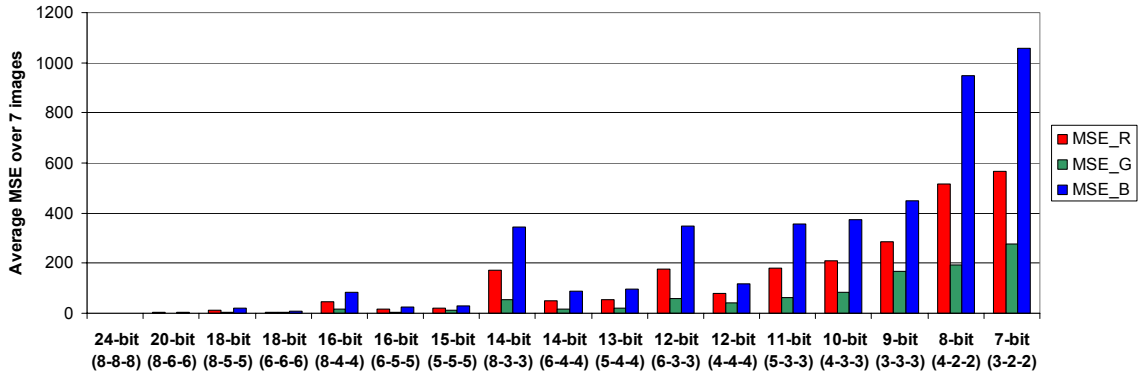


Figure 15. MSEs for various pixel word sizes. The form (n,m,l) represents n, m, and l bits for Y, Cb, and Cr, respectively.

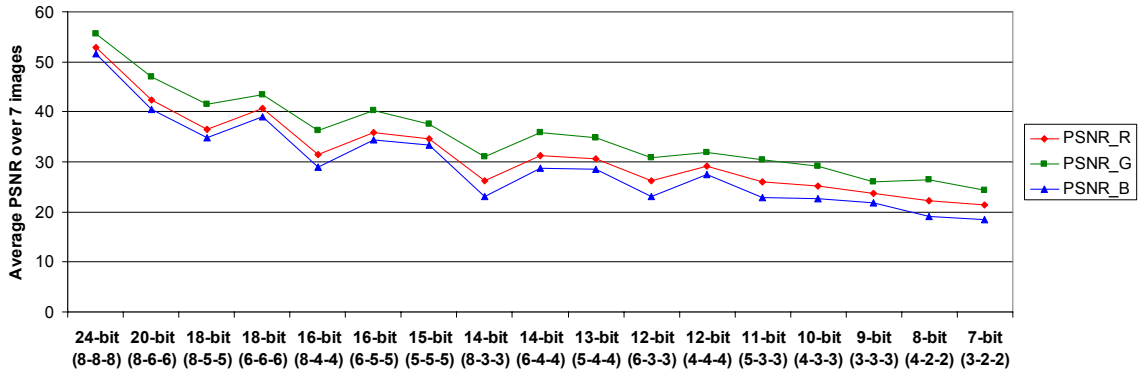


Figure 16. PSNRs for various pixel word sizes.

In addition to the quantitative evaluation, a qualitative evaluation must be done because a visual assessment of the processed image is the best subjective measure for determining the efficiency of the method. Figures 17, 18, and 19 show original images with converted output images for various pixel word sizes (available in color at [21]). As can be seen, the converted images having greater than or equal to five bits in all three channels provide satisfactory image quality. For those having less than five bits for at

least one color channel, however, significant image degradation occurs. Moreover, too much truncation affects contouring, making the image cartoon-like.

Figure 17. Original *Tank* image with converted output images for various pixel word sizes.

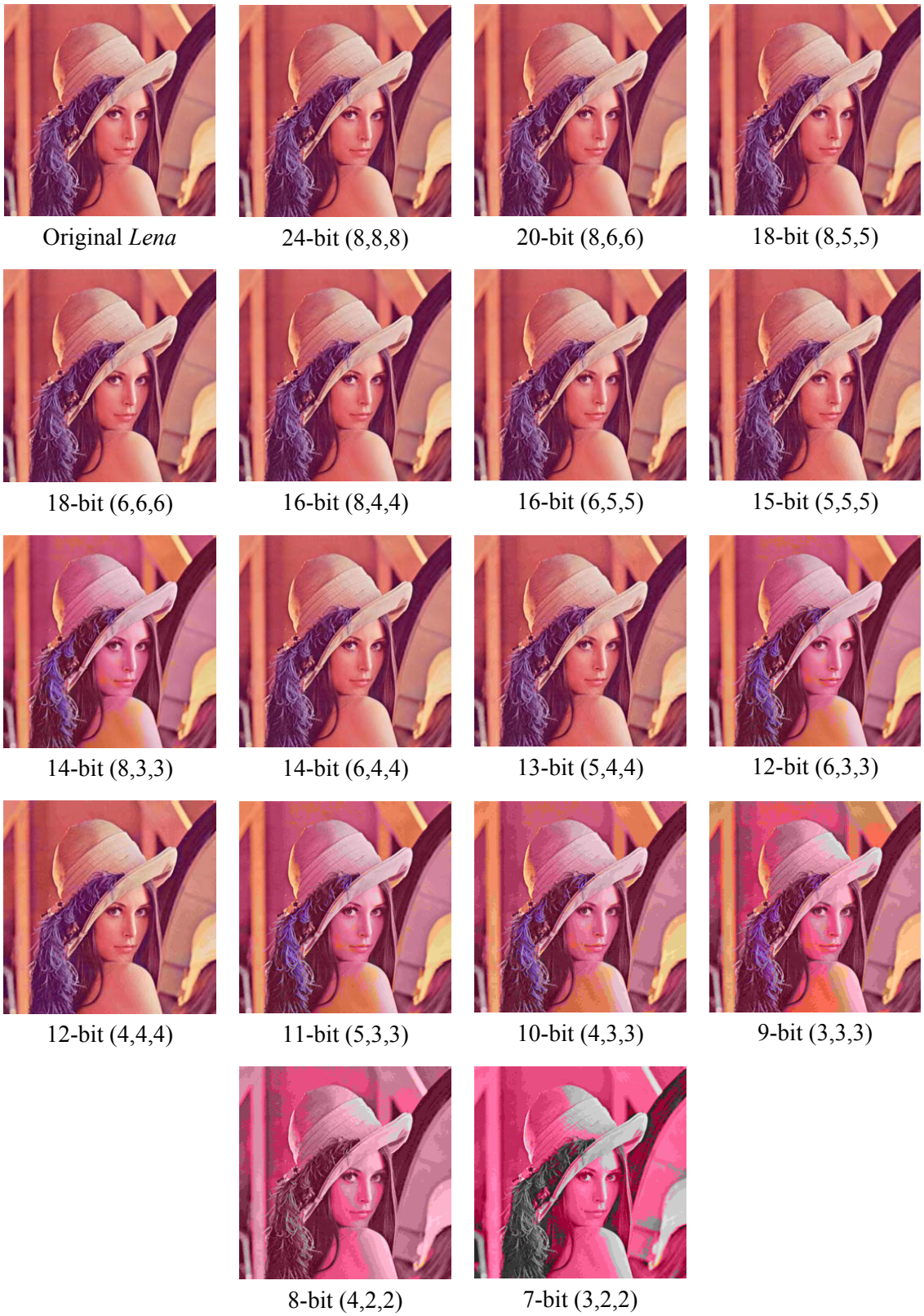


Figure 18. Original *Lena* image with converted output images for various pixel word sizes.

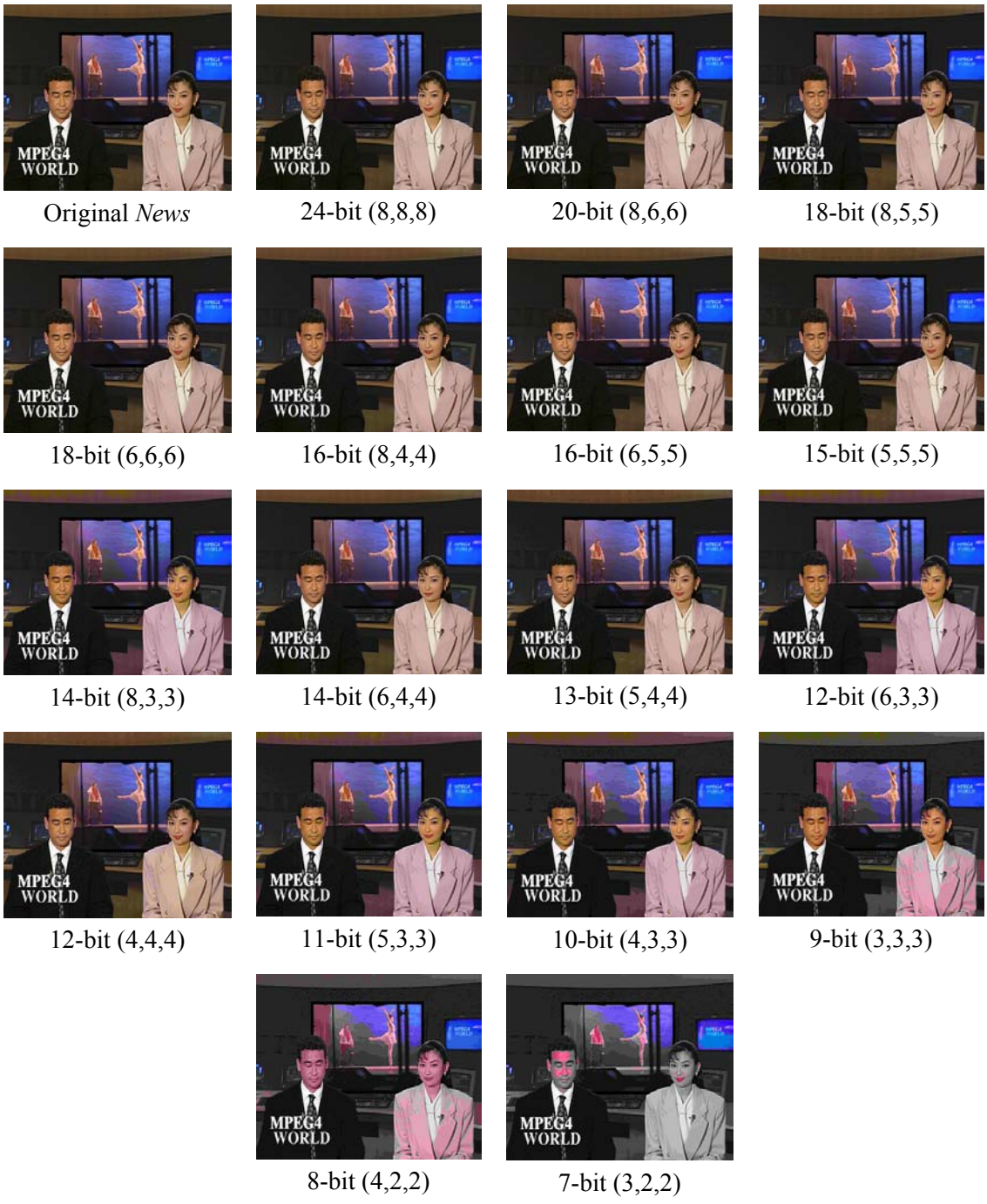


Figure 19. Original *News* frame with converted output images for various pixel word sizes.

Out of several acceptable color representations, the rest of this dissertation focuses on the 16-bit (6:5:5) YCbCr representation for reduced-memory, embedded video processing. The 16-bit YCbCr representation reduces the average per pixel word storage requirements by 33% over the 24-bit representation while maintaining acceptable PSNR performance. The next section evaluates the 16-bit YCbCr representation on motion estimation.

2.5.2 Motion Estimation using the 16-bit YCbCr Representation

The effectiveness of the 16-bit (6:5:5) YCbCr representation is evaluated using motion estimation (ME). In this experiment, the two implementations of ME are executed using MATLAB for a test suite of two color videos, each containing 40 frames of three-band CIF resolution (352×288) pixels. One implementation uses 24-bit YCbCr data, while the other uses 16-bit YCbCr data. In the experiment, a macroblock of 16×16 pixels and a search range of ±8 are considered.

Figures 20 and 21 show the PSNR values versus frame number for the 24- and 16-bit implementations of ME. The reported PSNR is the average PSNR of the three channels (e.g., RGB). Experimental results indicate that the overall quality of ME using the 16-bit YCbCr data format is comparable to the 24-bit YCbCr ME performance, indicating 30.9 dB versus 31.6 dB for the *Foreman* video and 32.2 dB versus 32.6 dB for the *News* video.

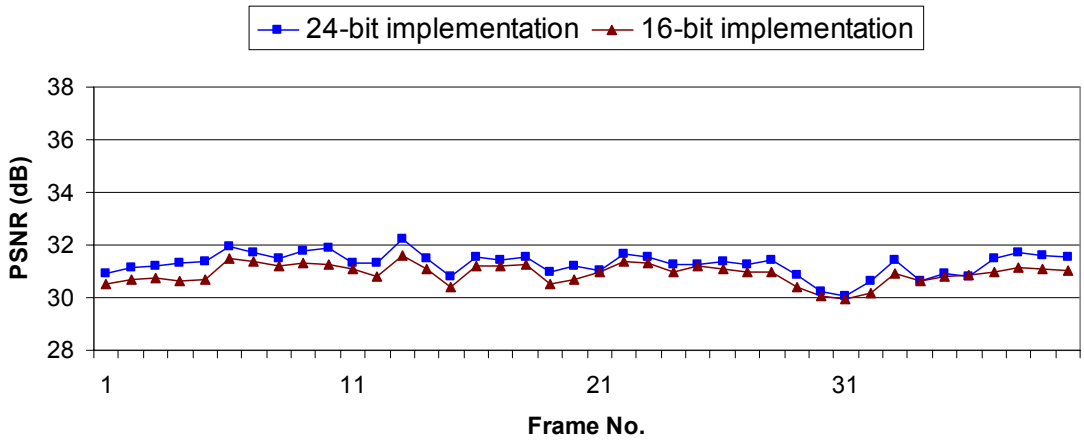


Figure 20. PSNR versus frame number for the *Foreman* video using motion estimation.

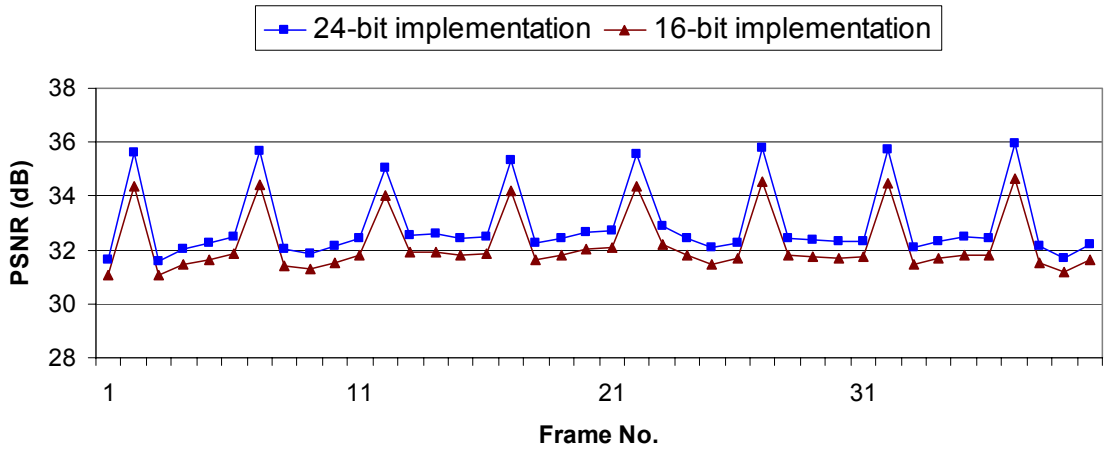


Figure 21. PSNR versus frame number for the *News* video using motion estimation.

The vector median filter (VMF) also has been examined with similar results, shown in Table 2 and Figure 22. In this experiment, each video frame was corrupted with a 4% impulse noise for each R, G, and B channel. In addition, this reduced pixel format is efficiently computed in an existing color converter without changing its circuitry, which is presented next.

Table 2. An average PSNR of the *Foreman* and *News* videos using the VMF.

	Foreman	*News*
16-bit VMF implementation	29.5 dB	31.7 dB
24-bit VMF implementation	29.7 dB	32 dB

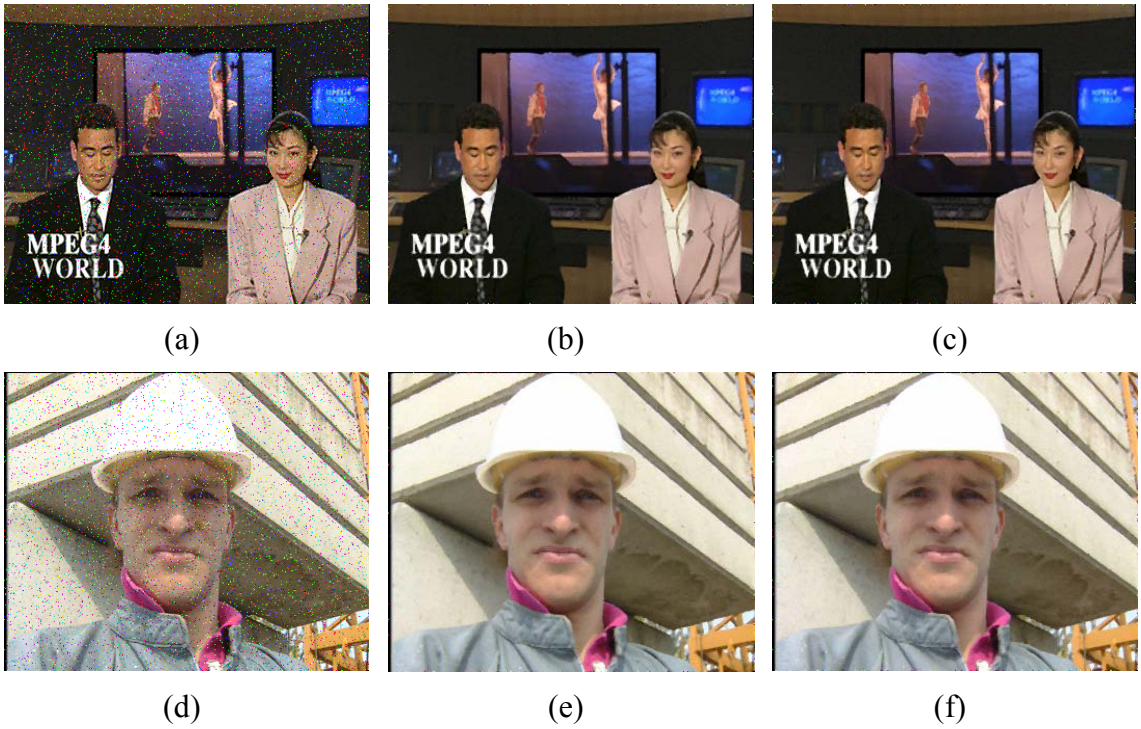


Figure 22. Corrupted images with recovered output images using the VMF (available in color at [21]): (a) and (d) 4% impulse noise; (b) and (e) the VMF for 24-bit YCbCr data; and (c) and (e) the VMF for 16-bit YCbCr data.

2.5.3 Implementation Costs

The 16-bit (6:5:5) YCbCr representation can be computed from 24-bit RGB pixel data using existing color conversion hardware. The conversion is defined in (14), where Y assumes values between [0, 63], and Cb and Cr assume values between [0, 31].

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.0748 & 0.1468 & 0.0285 \\ -0.0211 & -0.0414 & 0.0625 \\ 0.0625 & -0.0524 & -0.0101 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 16 \\ 16 \end{bmatrix} \tag{14}$$

This color transformation matrix can be computed with nine cycle latency and a three cycle per pixel throughput using the pipelined datapath, shown in Figure 23 (from [3]). For example, 0.0748 in the upper left-hand corner in (14) can be approximated by the sum $2^{-4}+2^{-7}+2^{-8}+2^{-11}$, and 0.0748R is represented by the sum R(4)+R(7)+R(8)+R(11), in which R(*n*) denotes a right shift of R by *n* bits. Following the same procedure, the 6-bit Y data can be obtained from

$$\begin{aligned} Y = {} & R(4) + R(7) + R(8) + R(11) + \\ & G(3) + G(6) + G(8) + G(9) + \\ & B(6) + B(7) + B(8) + B(10). \end{aligned} \tag{15}$$

The barrel shifter in Figure 23 then loads four data values at a time. For example, [R(4), R(7), R(8), R(11)] are loaded in the first cycle, [G(3), G(6), G(8), G(9)] are loaded in the second cycle, and so on. Using pipelining, a color pixel transformation can be completed every three cycles. To obtain the RGB values from a set of YCbCr values, the same hardware can also be used for the inverse matrix operation. Thus, the 24-bit RGB to 16-bit YCbCr (6:5:5) conversion can be computed in a simple datapath without the need for area intense multiplication hardware.

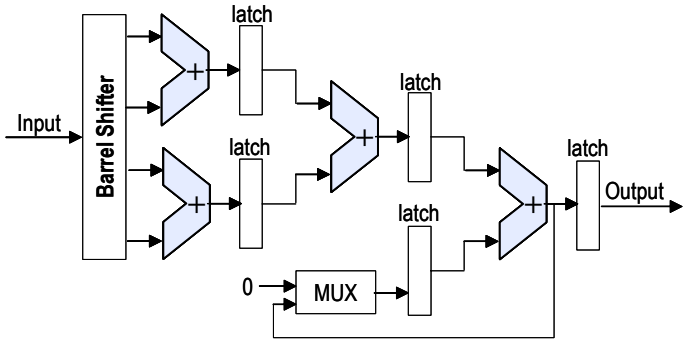


Figure 23. A block diagram of a color converter.

2.5.4 Other Benefits from the 16-bit YCbCr Representation

In addition to reducing pixel storage requirements, the 16-bit YCbCr representation is useful for an efficient color-aware instruction set (CAX) design. Employing this reduced pixel format, CAX supports parallel operations on two-packed 16-bit YCbCr data in a 32-bit datapath processor, shown in Figure 24, providing greater concurrency for processing color image sequences. Chapters 3 through 5 and Appendix A present the impact of CAX on processing performance and cost for color imaging applications in three major processor architectures: superscalar, very long instruction word (VLIW), and embedded single instruction, multiple data (SIMD) array processors.

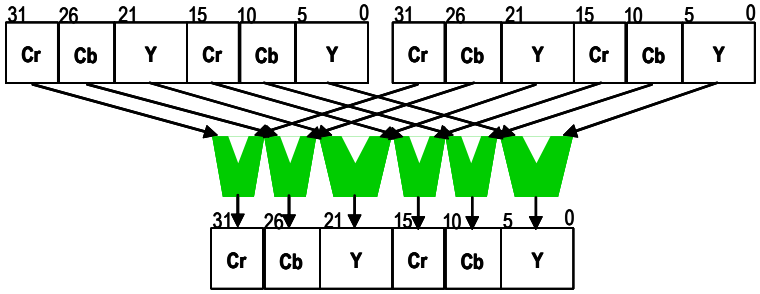


Figure 24. A 32-bit CAX operation.

2.6 Conclusion

This chapter has explored color imaging for multimedia to provide new opportunities to define an efficient architecture for embedded multimedia systems. Several color space models with varying subsampling factors have been evaluated to determine the most efficient color space that consistently reduces pixel information while maintaining image quality. The YCbCr space performs the best for all test images out of four well-known color space models since the human eye is less sensitive to chrominance channels. This chapter has also investigated the use of color information in multimedia applications using a vector approach. The vector approach improves the accuracy of the

process and overall image quality since it takes into account the correlation between color channels. Furthermore, several color representations with varying pixel word sizes have been evaluated to identify the most efficient representation in terms of storage requirements and color accuracy. In particular, a 16-bit (6:5:5) YCbCr representation has been examined for reduced-memory, embedded video processing. The 16-bit YCbCr representation reduces the average per pixel word storage requirements by 33% over the 24-bit YCbCr representation while maintaining acceptable PSNR performance. Moreover, employing this reduced pixel format, an efficient color-aware instruction set has been introduced that supports parallel operations on two-packed, quantized 16-bit YCbCr data in a 32-bit datapath processor, providing greater concurrency and efficiency for processing color image sequences. The next chapter presents the impact of CAX on processing performance and energy consumption for color imaging applications in superscalar ILP processors.

CHAPTER 3

UTILIZING COLOR SUBWORD PARALLELISM IN SUPERSCALAR ILP PROCESSORS

3.1 Introduction

As digital multimedia is rapidly revolutionizing our society, its applications, including color image and video processing, are becoming some of the dominant computing workloads [24]. These applications, however, demand tremendous computational and I/O throughput. The abundant data parallelism inherent to these applications has motivated the development of multimedia extensions on general-purpose processors (GPPs) to improve the performance of media-centric applications. Examples include Intel MMX™ [67], SSE™ and SSE-2 [70], Hewlett Packard MAX2 for the PA-RISC architecture [53], Sun VIS for SPARC [80], MIPS MDMX [60], Alpha MVI [75], and Motorola ALTIVEC for PowerPC™ architecture [63]. These extensions exploit subword parallelism by packing several small data elements (e.g., eight-bit pixels) into a single wide register (32-, 64-, or 128-bit) while processing these separate data elements in parallel without requiring extra registers or operations. While the improvement in performance has been exciting and encouraging, they poorly support the vector processing of color image sequences in which each pixel computation is simultaneously performed on 3-D YCbCr channels. In particular, their performance is limited in dealing with both color pixel data that are not aligned on boundaries that are powers of two (e.g., visually adjacent pixels from each band are spaced three bytes apart) and storage data types that are inappropriate for computation (necessitating conversion overhead before and usually following the computation) [77]. Although the band separated format (e.g.,

the red data for adjacent pixels are adjacent in memory) is the most convenient for single instruction, multiple data (SIMD) processing, a significant amount of overhead for data alignment is expected prior to SIMD processing. Even if the SIMD multimedia extensions store the pixel information as a packed 32-bit word composed of an eight-bit R, G, and B, and unused (U) field (band-interleaved format) in a 32-bit wide register, subword parallelism cannot be exploited on the operand of the unused field. Moreover, since the RGB space does not model the perceptual attributes of human vision well, the RGB to YCbCr conversion is necessary for further color image and video processing [85][36]. Although the SIMD multimedia extensions can handle the color conversion process in software, the hardware approach would be much more efficient.

A new color-aware instruction set extension (CAX) for superscalar instruction-level parallel (ILP) processors is presented to solve the problems inherent to RGB extensions by supporting parallel operations on two-packed 16-bit (6:5:5) YCbCr data in a 32-bit datapath processor. As illustrated in the previous chapter, the YCbCr space allows coding schemes that exploit the properties of human vision by truncating some of the less important data in every color pixel and allocating fewer bits to the high-frequency chrominance components that are perceptually less significant. Thus, the 16-bit YCbCr representation provides satisfactory image quality. In addition, CAX employs color-packed accumulators that provide a solution to overflow and other issues caused by packing data as tightly as possible by implicit width promotion and adequate space.

This chapter evaluates CAX in comparison to a representative multimedia extension, MDMX, an extension of MIPS. MDMX was chosen as a basis of comparison because it provides an effective way of dealing with reduction operations by using a wide

packed accumulator that successively accumulates the results produced by operations on multimedia vector registers. Other multimedia extensions poorly support vector processing in a 32-bit datapath processor without accumulators. To handle vector processing on a 64-bit or 128-bit datapath, they require frequent packing/unpacking of operand data, deteriorating their performance.

Experimental results show that CAX outperforms MDMX in speedup (3× to 5.8× with CAX, but only 1.6× to 3.2× with MDMX over the baseline performance) on the same dynamically scheduled, four-way issue superscalar processor. CAX also outperforms MDMX in energy reduction (68% to 83% reduction with CAX, but only 39% to 69% reduction with MDMX over the baseline version). Furthermore, CAX exhibits higher relative performance for low-issue rates. For example, CAX achieves an average speedup of 4.7× over the baseline 1-way issue performance, but 3× over the baseline 16-way issue performance. These results demonstrate that CAX is an ideal candidate for embedded multimedia systems in which high issue rates and out-of-order execution are too expensive.

Performance achieved by CAX is further enhanced through loop unrolling (LU) [26][86], an optimization technique that reorganizes and reschedules the loop body, which contains the most critical code segments for color imaging applications. In particular, LU reduces loop overhead while exposing ILP for machines with multiple functional units within the loops. Experimental results indicate that LU (by a factor of three for three programs and four for other programs) provides an additional 4%, 19%, and 21% performance improvement for the baseline, MDMX, and CAX versions,

respectively. These results suggest that the CAX plus LU technique has the potential to provide the higher performance required by emerging color imaging applications.

The rest of this chapter is organized as follows. Section 3.2 presents multimedia extensions to general-purpose processors along with research efforts using the multimedia extensions. Section 3.3 presents a summary of the CAX instruction set along with pictorial examples. Section 3.4 describes the selected color imaging applications, the modeled architectures, and the simulation methodology for the evaluation of CAX. Section 3.5 presents the experimental results and their analysis, and Section 3.6 concludes this chapter.

3.2 Related Research

3.2.1 Multimedia Extensions to General-Purpose Processors

Manufactures of general-purpose processors (GPPs) have included multimedia extensions to their instruction set architectures (ISAs) to support multimedia applications. The main idea in the extensions is exploiting subword parallelism within the context of a dynamically scheduled superscalar ILP machine. Table 3 shows the list of all major microprocessor vendors and shipped/announced multimedia instruction set extensions for their architectures [76]. These multimedia extensions support many instructions that enable simultaneous processing of several small data elements (e.g., eight-bit pixels) packed into a single wide register (e.g., 64-, or 128-bit). Depending on the target applications of a vendor, multimedia extensions vary widely. Motorola AltiVec has a large number of SIMD instructions (162 instructions), while HP MAX-1 has only a few (eight instructions). Many of the instruction sets, such as AMD 3DNow!, DEC MVI,

Intel MMX, and Sun VIS, are based on 64-bit wide registers, while Motorola AltiVec and Intel SSE are based on 128-bit wide registers. A notable exception is MIPS MDMX, which uses a single wide packed accumulator that successively accumulate the results produced by operations done with multimedia vector registers. Despite the similarities, each approach is unique. For example, MAX-2 reuses the integer registers and execution units while requiring virtually no additional execution hardware, but AltiVec requires an entirely new execution unit.

Table 3. Microprocessor multimedia extensions.

Processor	Extension	Product	Instructions	Register File
HP	MAX-1	1994	9	Integer (31x64b)
Sun	VIS	1995	121	FP (32x64b)
HP	MAX-2	1995	8	Integer (32x64b)
MIPS	MIPS-V	(-)	29	FP (32x64b)
MIPS	MDMX	(-)	74	FP (32x64b), Acc. (1x92b)
Intel	MMX	1997	57	FP (8x64b)
DEC	MVI	1997	13	Integer (31x64b)
Cyrix	Extended MMX	1997	12	FP (8x64b)
AMD	3D Now!	1998	21	FP (8x64b)
Intel	SSE	1999	70	8x128b
Motorola	AltiVec	1999	162	32x128b
MIPS	MIPS-3D	(-)	23	FP (32x64b)
AMD	Enhanced 3D Now!	1999	24	FP (8x64b)
Intel	SSE2	(-)	144	8x128b

Depending on the vendors, a multimedia instruction set extension contains some or all of the following instructions:

Modulo/Saturating

Modulo (or wraparound) arithmetic can produce partial results when overflow occurs, while saturating arithmetic clamps the output value to the largest or smallest

possible value for the given data type. Unlike modulo arithmetic, saturating arithmetic requires adding a little cost in the form of separate instructions for signed and unsigned operands because values must be interpreted by the hardware as a particular data type.

Parallel Compare Instructions

There are two types of parallel compare (*Pcmp*) instructions: an element mask and a bit mask. The element mask *Pcmp* instruction compares pairs of the sub-elements in the two source registers while generating either all 1s or all 0s for each sub-element comparison. The bit mask *Pcmp* instruction is similar, except that it generates either a one-bit true or false indicator for each sub-element comparison. Intel's MMX *pcmpeqw* instruction, for example, compares pairs of the packed 16-bit values in the two 64-bit source registers while generating either all 1s (0xffff) or 0s (0x0000) of each 16-bit sub-element for a 64-bit wide element mask. These masks are then used in conjunction with 64-bit logical operations, such as *AND*, *ANDN*, and *OR* to achieve the desired conditional assignment. On the other hand, Sun's VIS uses the bit mask *Pcmp* instruction to control the partial store instruction. Only sub-elements corresponding to a "1" bit in the bit mask are written to memory; other sub-elements remain unchanged.

Parallel Min/Max Instructions

Parallel min/max instructions output the minimum or maximum values of the corresponding elements in the two separate input registers, respectively.

Pack/Unpack Instructions

Pack instructions truncate larger sub-elements into smaller ones in tightly packed fields, while unpack instructions expand smaller sub-elements into larger ones. Figures

25(a) and (b) illustrate the packing of two registers into one register and the complementary operation of unpacking, respectively.

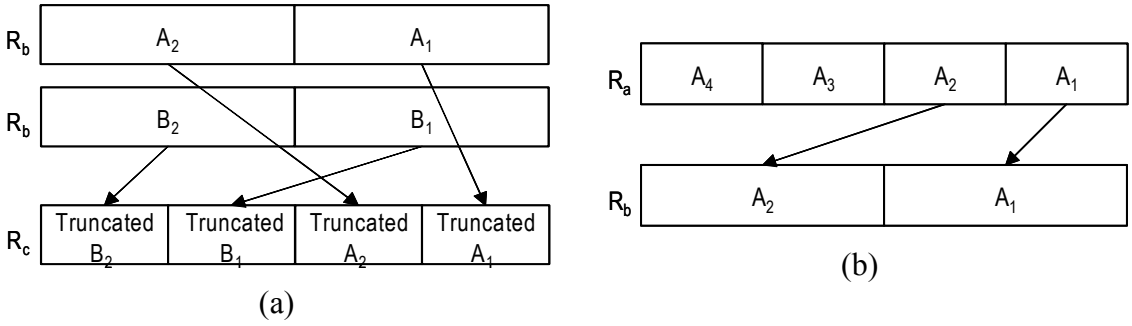


Figure 25. (a) A pack instruction. (b) An unpack instruction.

Permute/Mix Instructions

Permute instructions having one packed data-type source allow any permutation of the source quantities in the packed data-type destination, while mix instructions mix every other quantity of a packed data-type source register with the corresponding quantity from the second source register. Figures 26(a) and (b) show a permutation instruction in which one sub-element is repeated twice and a mix instruction, respectively.

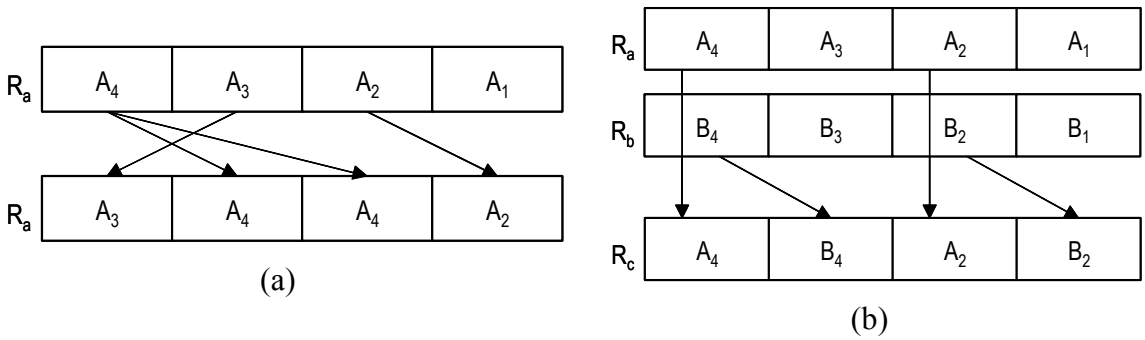


Figure 26. (a) A permute instruction. (b) A mix instruction.

Memory Instructions

A parallel load instruction can load multiple-packed elements into a register. A store instruction is similar, except that it stores into memory. All vendors include these

instructions. Moreover, since most multimedia computations have highly predictable memory access patterns, prefetching instructions are useful to reduce the number of cache miss penalties by fetching the cache block at a specified address into the cache from main memory if it is not already there.

Special-Purpose Instructions

Some vendors include special-purpose instructions that accelerate multimedia kernels. DEC's MVI, Sun's VIS, AMD's enhanced 3DNow!, and Intel's SSE, for example, include a sum of absolute differences (SAD) instruction that calculates the absolute differences of pairs of the sub-elements in the two source registers while summing all the differences in the destination register, as shown in Figure 27. The SAD instruction is commonly used in motion estimation for video compression [70][80]. In addition to the SAD instruction, Intel's SSE includes a packed average instruction that enables half-pixel interpolation in motion compensation by averaging a set of pixel values with pixels spatially offset by one, horizontally, vertically, or both [70]. On the other hand, AMD's 3DNow! includes reciprocal and square-root approximation instructions that typically have very high latency and are implemented as hardware lookup tables [66]. These instructions are used in 3-D rendering applications that use floating-point math functions.

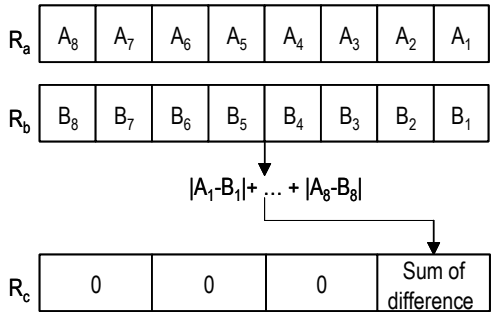


Figure 27. A SAD instruction.

The multimedia extensions have been such a success in general-purpose processors because they enhance the performance of multimedia applications with minimum hardware modification. For example, if the word size of a machine is 32 bits, the adder can be used to implement four eight-bit or two 16-bit additions in parallel by disconnecting the carry chain in the adder at every fourth or second position, respectively. The carry chain prevents an overflow of processing one subword datum into the next. A possible partitioned arithmetic logic unit (ALU) implementation is shown in Figure 28. Additional hardware is needed for the specific multimedia functions, but overall the typical area overhead for multimedia extensions in GPPs is only between 0.1% (HP's MAX-2) to 3% (Sun's VIS) of the entire processor die size [32].

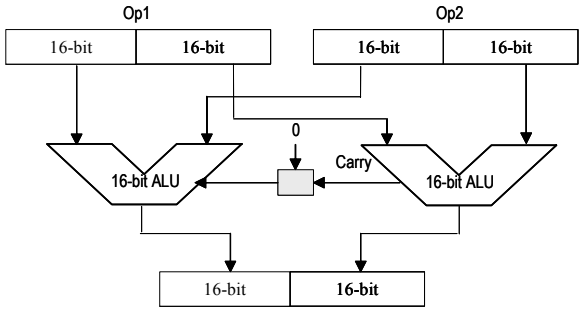


Figure 28. Partitioned ALU functional unit implementation.

The main disadvantage of using multimedia extensions is that no efficient compiler support is available for automating the multimedia extensions because of the lack of adequate high-level language constructs that utilize subword parallelism. In general, the partitioned ALU instructions are inserted into high-level language code manually by programmers in a form of intrinsic functions or assembly libraries provided by vendors.

3.2.2 Research Efforts Using Multimedia Extensions

Numerous groups and individuals have addressed the effectiveness of multimedia extensions for multimedia applications on general-purpose processors [63][77][6-52]. Bhargava *et al.* evaluated the multimedia instruction set extension (MMX) for a set of DSP and multimedia applications in the x86 architecture [6]. They observed that a finite impulse response (FIR) filter kernel showed a reasonable speedup of 1.57× (a 57% performance improvement) over the baseline performance because of the process of one input at a time. An infinite impulse response (IIR) filter kernel, however, showed a more impressive speedup of 2.55× due to block processing of the input samples, increasing data-level parallelism and reducing the number of functions called. In their study, image applications were the best suited for MMX because an image was stored in a large array of eight-bit data and properly aligned on eight-byte boundaries, showing a speedup of 5.5× and an 81% reduction in the dynamic instruction count.

Ranganathan *et al.* [71] evaluated the performance of image and video processing applications on an UltraSPARC processor with and without the VIS media extensions. They observed that a 4-way issue, out-of-order processor provided 2.3× to 4.2× performance improvement over a single-issue, in-order processor, and the VIS extensions provided an additional 1.1× to 4.2× performance improvement. In [51], Lappalainen *et al.* evaluated a video decoder on an Intel Pentium III with streaming SIMD extensions (SSE) and observed that an SSE-optimized video decoder provided a speedup of 3.41× over the baseline C version. In [63], Nguyen *et al.* evaluated the AltiVec technology on the PowerPC microprocessor in DSP and multimedia algorithms and observed that the

AltiVec technology provided a speedup ranging from 1.6× to 11.7× and 45% to 90% reductions in the dynamic instruction count.

Unlike the studies discussed above that have focused primarily on a single instruction set in isolation, Slingerland *et al.* conducted a thorough evaluation of the performance among five instruction sets on Berkeley multimedia benchmark kernels while comparing contemporary implementations of the multimedia ISA extensions with each other [77]. In [52], Lee presented an overview of three multimedia extensions, MAX for PA-RISC, MMX for ix86, and VIS for SPARC processor architectures.

Although many researchers have evaluated the performance of multimedia applications, the existing benchmark suites are still in their initial stage of development and do not include a variety of color imaging applications that are a large part of multimedia presentations. Since color imaging applications are simultaneously performed on 3-D color channels, they require more computational throughput. A color-aware instruction set extension (CAX) is presented next that improves the performance of color imaging applications.

3.3 A Color-Aware Multimedia Instruction Set for Color Imaging Applications

A color-aware instruction set (CAX) applied to current microprocessor ISAs targets the acceleration of color image and video processing applications. CAX supports parallel operations on two-packed 16-bit (6:5:5) YCbCr data in a 32-bit datapath processor, providing greater concurrency and efficiency for processing color image sequences. In addition, CAX employs color-packed accumulators that provide a solution to overflow and other issues caused by packing data as tightly as possible by implicit

width promotion and adequate space. Figure 29 illustrates three types of operations: (1) a baseline 32-bit operation, (2) a 4 × 8-bit SIMD operation used in many general-purpose processors, and (3) a 2 × 16-bit CAX operation employing heterogeneous (non-uniform) subword parallelism.

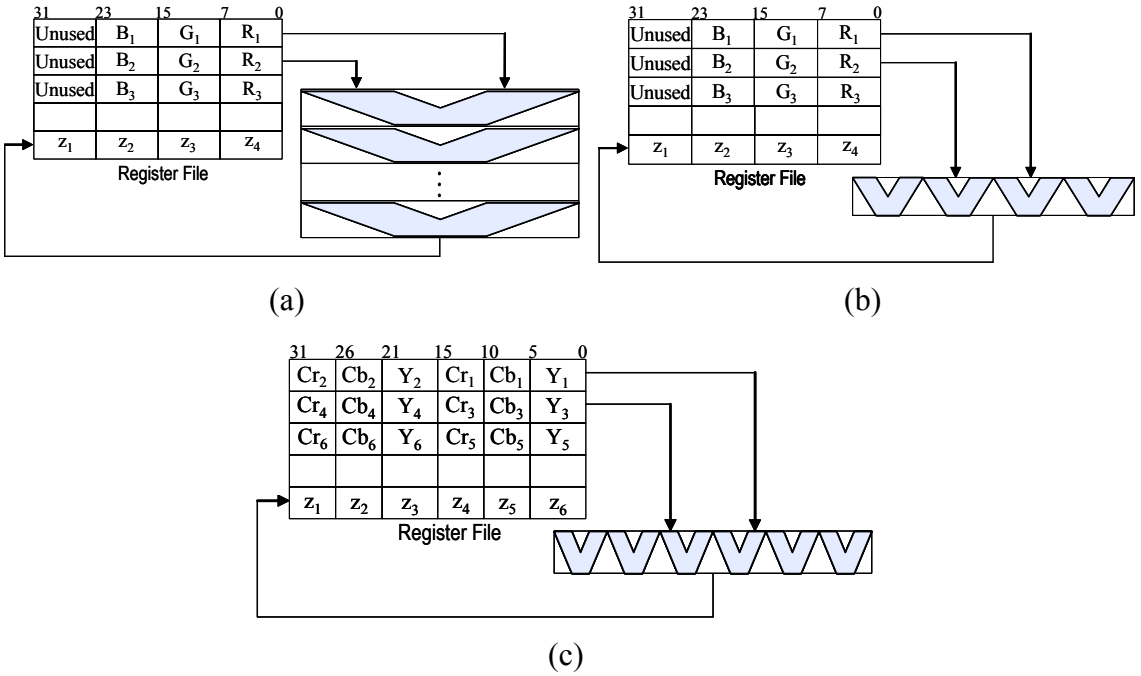


Figure 29. Types of operations: (a) a baseline 32-bit operation, (b) a 32-bit SIMD operation, and (c) a 32-bit CAX operation.

For color images, the band data may be interleaved (e.g., the red, green, and blue data of each pixel are adjacent in memory) or separated (e.g., the red data for adjacent pixels are adjacent in memory). The band separated format is the most convenient for SIMD processing, but a significant amount of overhead for data alignment is expected prior to SIMD processing. Moreover, traditional SIMD data communication operations have trouble with the band data that are not aligned on boundaries that are powers of two (e.g., adjacent pixels from each band are visually spaced three bytes apart) [77]. Even if the SIMD multimedia extensions store the pixel information in the band-interleaved

format (i.e., |R|G|B|Unused| in a 32-bit register), subword parallelism can not be exploited on the operand of the unused field. Furthermore, since the RGB color space does not model the perceptual attributes of human vision well, the RGB to YCbCr conversion is required prior to color image processing.

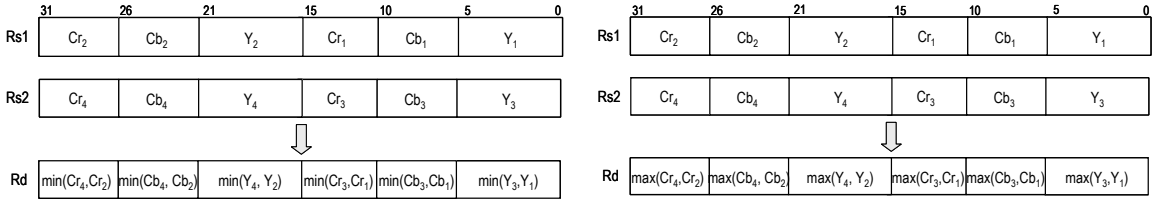
CAX solves problems inherent to packed RGB extensions by direct support for YCbCr data processing and a proper alignment of two-packed 16-bit data on 32-bit boundaries rather than depending solely on generic subword parallelism. The CAX instructions are classified into four different groups: (1) parallel arithmetic and logical instructions, (2) parallel compare instructions, (3) permute instructions, and (4) special-purpose instructions.

3.3.1 Parallel Arithmetic and Logical Instructions

Parallel arithmetic and logical instructions include packed versions of addition (ADD_CRCBY), subtraction (SUBTRACT_CRCBY), and averaging (AVERAGE_CRCBY). The addition and subtraction instructions include a saturation operation that clamps the output result to the largest or smallest value for the given data type when an overflow occurs. Saturating arithmetic is particularly useful in pixel-related operations, for example, to prevent a black pixel from becoming white if an overflow occurs. The packed average instruction is useful for blending algorithms, which takes two packed data types as input, adds corresponding data quantities, and divides each result by two while placing the result in the corresponding data location. The rounding is performed to ensure precision over repeated average instructions.

3.3.2 Parallel Compare Instructions

Parallel compare instructions include `CMPEQ_CRCBY`, `CMPNE_CRCBY`, `CMPGE_CRCBY`, `CMPGT_CRCBY`, `CMPLE_CRCBY`, `CMPLT_CRCBY`, `CMOV_CRCBY` (conditional move), `MIN_CRCBY`, and `MAX_CRCBY`. These instructions compare pairs of sub-elements (e.g., Y, Cb, and Cr) in the two source registers. Depending on the instructions, the results are varied for each sub-element comparison. The `CMPEQ_CRCBY` instruction, for example, compares pairs of sub-elements in the two source registers while writing a bit string of 1s for true comparison results and 0s for false comparison results to the target register. The first seven instructions are useful for a condition query performed on the incoming data such as chroma-keying [68]. The last two instructions, `MIN_CRCBY` and `MAX_CRCBY`, are especially useful for median filtering, which compare pairs of sub-elements in the two source registers while outputting the minimum and maximum values to the target register, respectively, shown in Figure 30.



(a) (b)

Figure 30. (a) A packed min instruction. (b) A packed max instruction.

3.3.3 Permute Instructions

Permute instructions include `MIX_CRCBY`, and `ROTATE_CRCBY`. These instructions are used to rearrange the order of quantities in the packed data type. The mix instruction mixes the sub-elements of the two source registers into the operands of the target register, and the rotate instruction rotates the sub-elements to the right by an

immediate value. Figures 31(a) and (b) illustrate the rotate and mix instructions, respectively, which are useful for performing a vector pixel transposition or a matrix transposition [78].

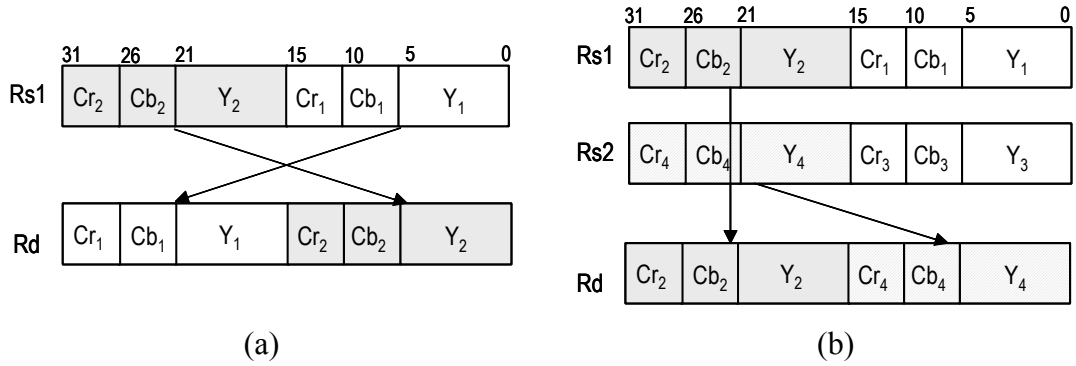


Figure 31. (a) A rotate instruction. (b) A mix instruction.

3.3.4 Special-Purpose Instructions

Special-purpose CAX instructions include `ADACC_CRCBY` (absolute-differences-accumulate), `MACC_CRCBY` (multiply-accumulate), `RAC` (read accumulator), and `ZACC` (zero accumulator), which provide the most computational benefits of all the CAX instructions. The `ADACC_CRCBY` instruction, for example, is frequently used in a number of algorithms for motion estimation. It calculates the absolute differences of pairs of sub-elements in the two source registers while accumulating each result in the packed accumulator, shown in Figure 32. The `MACC_CRCBY` instruction is useful in DSP algorithms that involve computing a vector dot-product, such as digital filtering and convolutions. The latter two instructions, `RAC` and `ZACC`, are related to the managing of the CAX accumulator.

These CAX instructions are included in the ISA of a dynamically scheduled superscalar processor to improve the performance of color imaging applications.

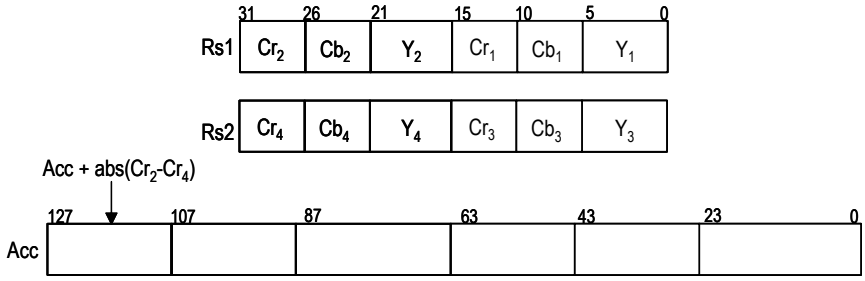


Figure 32. An absolute-differences-accumulate instruction.

3.4 Methodology

This section describes the selected color imaging applications, the modeled architectures and tools, and a methodology infrastructure to evaluate the CAX instruction set.

3.4.1 Color Imaging Applications

We study five imaging applications to capture a range of color imaging for multimedia: color edge detection using a vector Sobel operator (VSobel), the scalar median filter (SMF), the vector median filter (VMF), vector quantization (VQ), and the full-search vector BMA (FSVBMA) of motion estimation within the MPEG standard. Although the SMF is not an example of vector processing, this study includes the SMF in the application suite because of its useful and well-known sorting algorithm. These applications, briefly summarized in Table 4 and introduced in Section 2.4, form significant components of many current and future real-world workloads such as streaming video across the internet, real-time video enhancement and analysis, and scene-visualization. All the applications are executed with CIF resolution (352×288) 3-band (i.e., channel) input image sequences.

Table 4. Summary of the benchmarks used in this study.

Application	Description
VSobel	Extracts color edge information from an image through a Sobel operator that accounts for local changes in both luminance and chrominance components.
SMF	Removes impulse noise from an image by replacing each color component with a median value in a 3 x 3 window that is moved across the entire image. The three resulting images are then combined to produce a final output image.
VMF	Suppresses impulse noise from an image through a vector approach that is performed simultaneously on three color components (i.e., Y, Cb, and Cr).
VQ	Compresses and quantizes collections of input data by mapping k-dimensional vectors in vector space $\mathbf{R}^k$ into a finite set of vectors [35]. A full search vector quantization using both the luminance and chrominance components is used to find the best match in terms of the chosen cost function.
FSVBMA	Removes temporal redundancies between video frames in MPEG/H.26L video applications. A full search block-matching algorithm using both the luminance and chrominance components is used to find one motion vector for all components.

3.4.2 Modeled Architectures and Tools

Figure 33 shows a methodology framework for this study. The Simplescalar-based toolset [2], an infrastructure for out-of-order superscalar modeling, is used to simulate a superscalar processor with and without MDMX or CAX, in which MDMX and CAX instructions are synthesized using annotations in the assembly files. The MDMX and CAX versions of the programs are generated by identifying the most time-consuming kernels by profiling and manually replacing the fragments of the baseline assembly language with ones containing MDMX and CAX instructions. Since the target platform is an embedded system, operating system interface code (e.g., file system access) is not included in this study. (Of course, the speedups of MDMX and CAX for complete programs may be less impressive than those for kernels due to Amdahl's Law [38].) In addition, all the implementations exclude the color conversion process. In other words, this study assumes that the baseline, MDMX, and CAX versions directly support YCbCr

data in the same general data format (e.g., |Unused|Cr|Cb|Y| for baseline and MDMX and |Cr|Cb|Y|Cr|Cb|Y| for CAX). Moreover, a fair approximation of MDMX is added to the PISA of the Simplescalar simulator. For example, MDMX is extended with additional instructions such as absolute-differences-accumulation or parallel-conditional-move in CAX. Thus, MDMX (containing 30 instructions) has similar instructions as CAX (containing 34 instructions) except for the permute instructions.

The Wattch-based simulator [10], an architectural-level power modeling, is also used to estimate energy consumption in each case. For the power estimates of the MDMX and CAX functional units (FUs), Verilog models for the baseline, MDMX, and CAX FUs are implemented and synthesized with the Synopsys design compiler (DC) using a 0.18-micron standard cell library. The reported power specifications of the MDMX and CAX FUs, shown in Table 5, are then normalized to the baseline FU, and the normalized numbers are applied to the Wattch simulator to determine the dynamic power for the superscalar processor with MDMX or CAX.

Table 5. Dynamic power estimates for 32-bit FU designs with 1GHz at operating voltage of 1.62.

	ALU	MAC
Baseline	12.5 mW	262.2 mW
MDMX	15.0 mW	305.2 mW
CAX	18.8 mW	299.9 mW

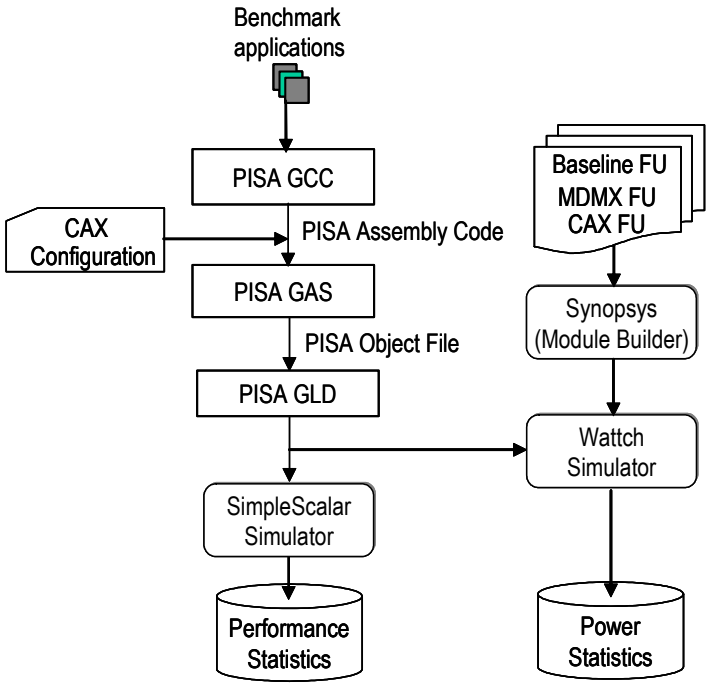


Figure 33. A methodology framework for dynamically scheduled simulations.

Table 6 summarizes the processor configurations used in this study. A wide range of superscalar processors is simulated by varying the issue width from 1 to 16 instructions per cycle and the instruction window size from 16 to 256. When the issue width is doubled, the number of functional units, load/store queues, and main memory widths are scaled accordingly, in which the L1 cache (instruction and data) and the L2 cache are fixed at 16 KB and 256 KB, respectively. This study assumes that both MDMX and CAX use two logical accumulators, and all the implementations are simulated with a 180 nm process technology at 600 MHz and aggressive, non-ideal conditional clocking. (Power is scaled linearly with port or unit usage, and unused units are estimated to dissipate 10% of the maximum power.) With these processor configurations, the next section evaluates the impact of CAX on processing performance and energy consumption for the selected color imaging applications.

Table 6. Processor configurations.

Parameter	1-way	2-way	4-way	8-way	16-way
Fetch/decode/issue/commit width	1	2	4	8	16
intALU/intMUL/fpALU/fpMUL/Mem	1/1/1/1/1	2/1/1/1/2	4/2/2/1/4	8/4/2/1/8	16/8/4/1/16
RUU (window) size	16	32	64	128	256
LSQ (Load Store Queue)	8	16	32	64	128
Main memory width	32 bits	64 bits	128 bits	256 bits	256 bits
Branch Predictor	Combined predictor (1 K entries) of bimodal predictor (4 K entries) table and 2-level predictor (2-bit counters and 10-bit global history)				
L1 D-cache	128-set, 4-way, 32-byte line, LRU, 1-cycle hit, total of 16 KB				
L1 I-cache	512-set, direct-mapped 32-byte line, LRU, 1-cycle hit, total of 16 KB				
L2 unified cache	1024-set, 4-way, 64-byte line, LRU, 6-cycle hit, total of 256 KB				
Main memory latency	50 cycles for first chunk, 2 thereafter				
Instruction TLB	16-way, 4096 byte page, 4-way, LRU, 30 cycle miss penalty				
Data TLB	32-way, 4096 byte page, 4-way, LRU, 30 cycle miss penalty				

3.5 Experimental Results

In the experiment, the three different versions of the programs are coded and simulated using the Simplescalar-based simulator for the evaluation of CAX: (1) baseline ISA without subword parallelism, (2) baseline plus MDMX ISA, and (3) baseline plus CAX ISA. The three different versions of each program have the same parameters, data sets, and calling sequences. In addition, the Wattch-based power simulator is used to evaluate the energy consumption of each benchmark. The dynamic instruction count, execution cycle count, and energy consumption of each case form the basis of the comparative study.

3.5.1 Performance-Related Evaluation Results

This section presents the impact of CAX on execution performance for the benchmarks. The effect of loop unrolling for each program is also presented.

3.5.1.1 Overall Results

Figure 34 illustrates execution performance (speedup in executed cycles) for different wide superscalar processors with MDMX and CAX, normalized to the baseline performance without subword parallelism. The results indicate that CAX outperforms MDMX for all the programs in terms of speedup. For the 4-way issue machine, for example, CAX achieves a speedup ranging from 3× to 5.8× over the baseline performance, while MDMX achieves a speedup ranging from only 1.6× to 3.2× over the baseline.

An interesting observation is that CAX exhibits higher relative performance for low-issue rates. For example, CAX achieves an average speedup of 4.7× over the baseline 1-way issue performance, but 3× over the baseline 16-way issue performance. This result demonstrates that CAX is an ideal candidate for embedded multimedia systems in which high issue rates and out-of-order execution are too expensive.

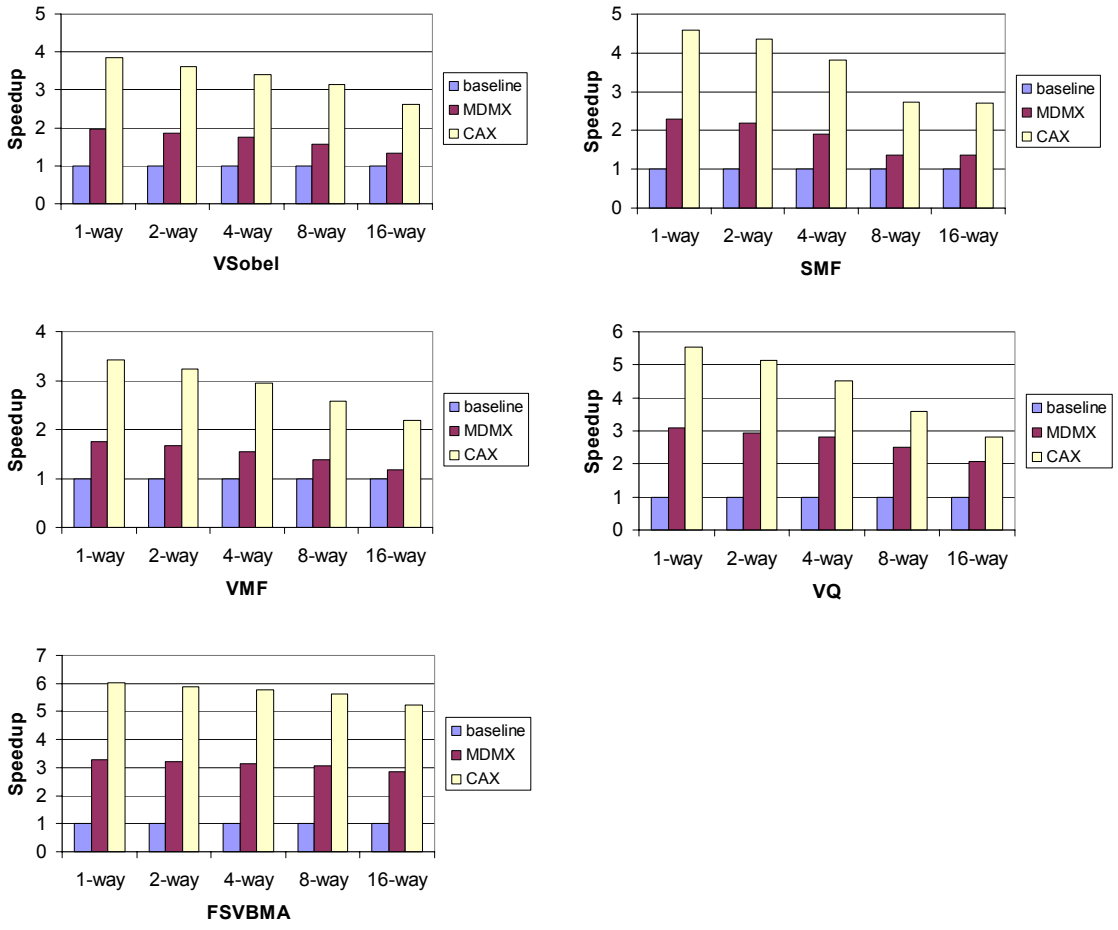


Figure 34. Speedups for different issue-rate processors with MDMX and CAX, normalized to the baseline performance.

3.5.1.2 Benefits from CAX

Figure 35 presents the distribution of dynamic instructions for the 4-way out-of-order processor with MDMX and CAX, normalized to the baseline version. Each bar divides the instructions into the functional unit (FU, combines ALU and FPU), control, memory, MDMX, and CAX categories. The use of CAX provides a significant reduction in the dynamic instruction count across all the programs.

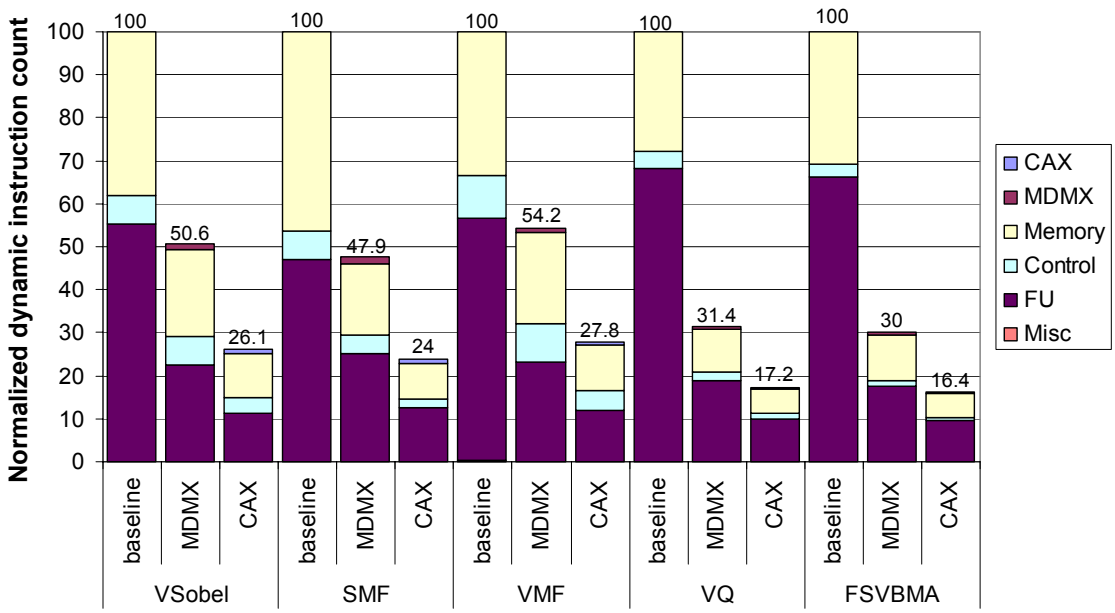


Figure 35. Impact of CAX on the dynamic (retired) instruction count.

Reductions in FU Instructions. The CAX arithmetic and logical instructions allow multiple arithmetic and logical instructions (typically three by processing three channels simultaneously) in addition to multiple iterations (typically two by processing two-packed YCbCr data) with one CAX instruction. Because of this property, all the programs using CAX reduce a significant number of the FU instructions and loop overhead, which increments or decrements index and address values. The reduction of the loop overhead further reduces the FU instruction count. Experimental results indicate that the FU instruction count decreases 73% to 86% (an average of 81%) with CAX, but only 47% to 73% (an average of 64%) with MDMX over the baseline version.

Reductions in Control Instructions. The CAX compare instructions allow multiple conditional (or branch) instructions with one equivalent CAX instruction, resulting in a large reduction in the control instruction count for all the programs. The control

instruction count decreases 47% to 76% (an average of 60%) with CAX, but only 2% to 57% (an average of 26%) with MDMX over the baseline version.

Reductions in Memory Instructions. With CAX, multiple packed data are transported from/to memory rather than individual components. CAX accumulator instructions (e.g., `MACC_CRCBY` and `ADACC_CRCBY`) further eliminate memory operations since immediate results are stored in the accumulator rather than in memory. Experimental results indicate that the memory instruction count decreases 68% to 83% (an average of 78%) with CAX, but only 37% to 66% (an average of 57%) with MDMX over the baseline version.

Overall, CAX clearly outperforms MDMX in consistently reducing the number of dynamic instructions required for each program. Performance improved by CAX can be further enhanced through loop unrolling, which is presented next.

3.5.1.3 Benefits from Loop Unrolling

Loop unrolling (LU) is a well-known optimization technique that reorganizes and reschedules the loop body. Since loops contain the most critical code segments for color imaging applications, LU achieves a higher degree of performance by reducing loop overhead and exposing instruction-level parallelism (ILP) for machines with multiple functional units within the loops. Thus, the LU plus CAX technique may provide the much higher degrees of parallelism and performance. Figures 36(a), (b), and (c) present an example of the inner loop of the BMA for vector quantization, the code after loop unrolling, and the loop from the perspective of CAX-level parallelism, respectively. The original loop is unrolled and reorganized through LU, shown in Figure 36(b). In the unrolled statement, multiple operands are then packed in each register with CAX, as

shown in the dotted-line boxes in Figure 36(c). CAX then replaces the fragments of the assembly language for isomorphic statements grouped together in the dashed-line boxes with ones containing CAX instructions. Since operands are effectively pre-packed in memory, they do not need to be unpacked when processed in registers. In particular, the LU plus CAX technique provides the following benefits:

- it reduces branch and address generation overhead,
- it reduces register pressure and memory traffic by transporting multiple packed data from a register to memory and vice versa, and
- it reduces a significant number of dynamic instruction counts.

```
for (i=0; i<4; i++) {
   sum_y   += abs( IV_Y[i]   – CV_Y[i]);
   sum_Cb  += abs( IV_Cb[i]  – CV_Cb[i]);
   sum_Cr  += abs( IV_Cr[i]  – CV_Cr[i]);
}
```

(a)

```
sum_y   += abs( IV_Y[i+0]   – CV_Y[i+0]);
sum_Cb  += abs( IV_Cb[i+0]  – CV_Cb[i+0]);
sum_Cr  += abs( IV_Cr[i+0]  – CV_Cr[i+0]);
sum_Y   += abs( IV_Y[i+1]   – CV_Y[i+1]);
sum_Cb  += abs( IV_Cb[i+1]  – CV_Cb[i+1]);
sum_Cr  += abs( IV_Cr[i+1]  – CV_Cr[i+1]);
sum_y   += abs( IV_Y[i+2]   – CV_Y[i+2]);
sum_Cb  += abs( IV_Cb[i+2]  – CV_Cb[i+2]);
sum_Cr  += abs( IV_Cr[i+2]  – CV_Cr[i+2]);
sum_Y   += abs( IV_Y[i+3]   – CV_Y[i+3]);
sum_Cb  += abs( IV_Cb[i+3]  – CV_Cb[i+3]);
sum_Cr  += abs( IV_Cr[i+3]  – CV_Cr[i+3]);
```

(b)

```
sum_y   += abs( IV_Y[i+0]   – CV_Y[i+0]);
sum_Cb  += abs( IV_Cb[i+0]  – CV_Cb[i+0]);
sum_Cr  += abs( IV_Cr[i+0]  – CV_Cr[i+0]);
sum_Y   += abs( IV_Y[i+1]   – CV_Y[i+1]);
sum_Cb  += abs( IV_Cb[i+1]  – CV_Cb[i+1]);
sum_Cr  += abs( IV_Cr[i+1]  – CV_Cr[i+1]);
sum_Y   += abs( IV_Y[i+2]   – CV_Y[i+2]);
sum_Cb  += abs( IV_Cb[i+2]  – CV_Cb[i+2]);
sum_Cr  += abs( IV_Cr[i+2]  – CV_Cr[i+2]);
sum_Y   += abs( IV_Y[i+3]   – CV_Y[i+3]);
sum_Cb  += abs( IV_Cb[i+3]  – CV_Cb[i+3]);
sum_Cr  += abs( IV_Cr[i+3]  – CV_Cr[i+3]);
```

(c)

Figure 36. (a) Original loop. (b) After loop unrolling. (C) CAX-level parallelism exposed after loop unrolling. IV and CV stand for the image vector and the codebook vector, respectively.

Table 7 presents speedups for the baseline, MDMX, and CAX versions with LU, normalized to those without LU, in which the VSobel, SMF, and VMF programs were

unrolled by a factor of three; others were unrolled by a factor of four. LU tends to be more effective for the CAX version than the baseline and MDMX versions, indicating 21%, 4%, and 19% performance gains in the CAX, baseline, and MDMX versions, respectively. One of the major reasons is that LU reduces a similar number of loop overhead instructions for all three versions, but the total number of executed instructions for the CAX version is smaller than that for the baseline or MDMX versions. The next section presents energy-related performance since energy is as critical for embedded multimedia systems as performance.

Table 7. Speedups of the baseline, MDMX, and CAX versions with LU, normalized to those without LU.

	VSobel	SMF	VMF	VQ	FSVBMA	Average
Baseline plus LU	1.05	1.06	1.07	1.04	1.02	1.04
MDMX plus LU	1.24	1.23	1.28	1.14	1.09	1.19
CAX plus LU	1.27	1.24	1.29	1.16	1.10	1.21

3.5.2 Energy-Related Evaluation Results

Figure 37 presents the distribution of energy consumption for the 4-way out-of-order processor with MDMX and CAX, normalized to the baseline version. Each bar divides the energy consumption into the cache, ALU, clock, window, and others (combines branch prediction, rename, load-store queue, and result bus) categories. When execution platforms employ identical clock rates, implementation technologies, and processor parameters, a shorter execution time results in lower energy consumption [79]. Thus, CAX reduces a large amount of total energy consumption for all the programs because of a significant reduction in the executed cycle count. Experimental results indicate that CAX reduces energy consumption from 68% (VMF) to 83% (FSVBMA) over the baseline. This is in contrast to MDMX, which reduces energy consumption from

only 39% (VMF) to 69% (FSVBMA) over the baseline. Since CAX reduces a large number of the ALUs, branches, and cache accesses, less energy is spent on the speculative execution and cache access units.

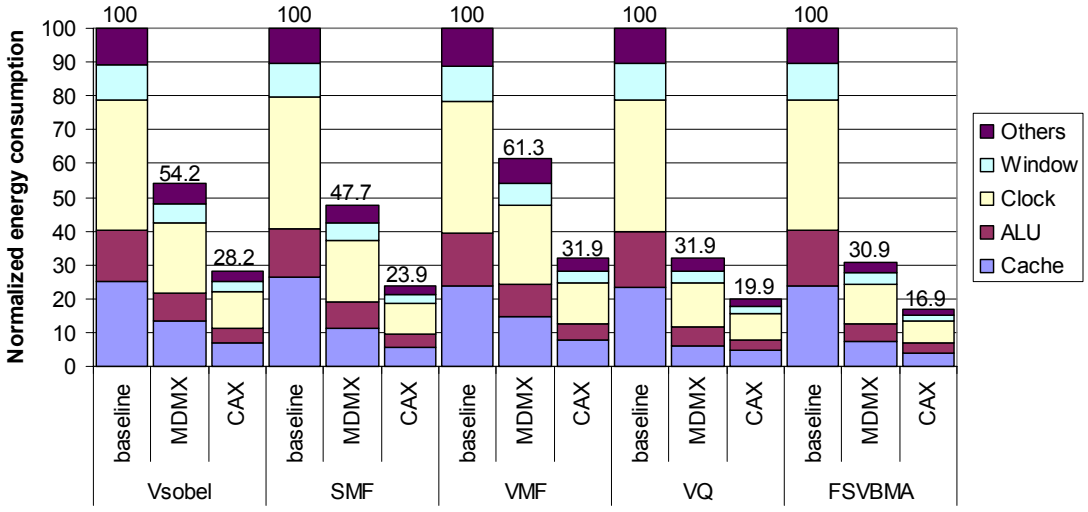


Figure 37. Impact of CAX on energy consumption.

The energy consumption is further reduced with LU for all three versions of the programs, showing an average energy reduction of 4.8%, 18.8%, and 19.2% for the baseline, MDMX, and CAX versions, respectively. In particular, LU reduces a large percentage of the power dissipation in the branch prediction hardware because it efficiently reduces branch overhead, indicating an energy reduction of 14.4%, 35.9%, and 36.3% in the branch prediction hardware for the baseline, MDMX, and CAX versions, respectively. Removing branches using LU also reduces the power dissipation in the fetch unit. The fetch unit fetches large basic blocks without being interrupted by taken branches, providing more work for the renaming unit and filling up the register update unit (RUU) faster. Thus, when the instruction queue and RUU are full, the fetch unit is stalled during the cycles. Because of this, the power dissipation of the fetch unit is

reduced. Clearly, LU is effective at reducing additional energy consumption for image processing kernels where loop overhead is significant.

3.6 Conclusion

A new color-aware multimedia extension (CAX) for dynamically scheduled ILP processors has been presented that improves the performance of color imaging applications. Harnessing parallelism within the human perceptual color space (e.g., YCbCr), CAX supports parallel operations on two-packed, quantized 16-bit YCbCr data in a 32-bit datapath processor, providing greater concurrency and efficiency for processing color image sequences. The key findings are the following:

- CAX outperforms MDMX (a representative MIPS multimedia extension) in speedup (3× to 5.8× with CAX, but only 1.6× to 3.2× with MDMX over the baseline performance) on the same dynamically scheduled, 4-way issue processor.
- CAX also outperforms MDMX in energy reduction (68% to 83% reduction with CAX, but only 39% to 69% with MDMX over the baseline version).
- Moreover, CAX exhibits higher relative performance for low-issue rates. For example, CAX achieves an average speedup of 4.7× over the baseline 1-way issue performance, but 3× over the baseline 16-way issue performance). These results demonstrate that CAX is an ideal candidate for embedded multimedia systems in which high issue rates and out-of-order execution are too expensive.

- Performance improved by CAX has been further enhanced through loop unrolling. LU provides an additional performance gain of 21%, 4%, and 19% for the CAX, baseline, and MDMX versions, respectively. These results demonstrate that the CAX plus LU technique has the potential to provide the higher degrees of performance required by emerging color imaging applications.

The effectiveness of CAX will be much more obvious in application-specific embedded systems (e.g., embedded SIMD arrays) that aim at providing sufficient computational power for specific applications but impose strict constraints on implementation chip area and energy consumption. This is because CAX benefits from greater concurrency as well as reduced pixel word storage (buffers, registers, and memory) that consumes a large percentage of silicon area. The next chapter presents the impact of CAX on processing performance and on both area and energy efficiency on a representative SIMD array architecture.

CHAPTER 4

IMPLEMENTATION AND EVALUATION OF THE COLOR-AWARE INSTRUCTION SET FOR LOW-MEMORY, EMBEDDED VIDEO PROCESSING IN DATA PARALLEL ARCHITECTURES

4.1 Introduction

Portable multimedia applications demand tremendous instruction throughput with a small area and limited energy available in a battery. Application-specific integrated circuits (ASICs) can meet the needed performance and cost goals for such portable multimedia systems. However, they provide limited, if any, programmability or flexibility necessary for varied application requirements.

General-purpose microprocessors (GPPs) offer the necessary flexibility and inexpensive processing elements, and multimedia extensions to GPPs have improved the performance of multimedia applications with little added cost to the processors. The designers of digital signal processors (DSPs) such as the Texas Instruments TMS320C64x families [82] and the Analog Devices TigerSharc processor [31] have followed the trend. However, despite some performance improvements through multimedia extensions, neither GPPs nor DSPs will be able to meet the much higher levels of performance required by emerging multimedia applications on higher resolution images. This is because they lack the ability to exploit the full data parallelism available in these applications.

Among many computationally efficient models available for imaging applications, single instruction, multiple data (SIMD) arrays are promising candidates for application-specific embedded multimedia systems because they replicate the datapath, data memory, and I/O to provide high processing performance with low node cost. Whereas instruction-

level or thread-level processors use silicon area for large multiported register files, large caches, and deeply pipelined functional units, SIMD arrays contain many more simple processing elements (PEs) for the same silicon area. As a result, SIMD arrays often employ thousands of PEs while possibly distributing and co-locating PEs with the data I/O to minimize storage and data communication requirements. The SIMD Pixel (SIMPil) processor [13][34][8] being developed at Georgia Tech, for example, is a low memory, monolithically integrated SIMD architecture that efficiently exploits massive data parallelism inherent in imaging applications while reducing data movement through a processing-in-place technique in which image data are directly transported into the PEs and stored there. While 2-D SIMD arrays, including SIMPil, are well suited for many imaging tasks that require processing of pixel data with respect to either nearest-neighbor or other 2-D patterns exhibiting locality or regularity, they are less amenable to the vector processing of color image sequences, in which each pixel computation is simultaneously performed on 3-D YCbCr channels. More specifically, since the 3-D vector computation is performed within innermost loops, its performance does not scale with larger PE arrays.

This chapter presents the CAX instruction set for such SIMD arrays as a solution to this performance limitation by supporting two-packed 16-bit YCbCr data in a 32-bit wide register, while processing this separate color data in parallel. In addition to greater concurrency, the ability to reduce data format size drastically reduces system cost. The reduction in data bandwidth also simplifies system design.

Experimental results using application simulation and technology modeling indicate that CAX outperforms MDMX across all the selected programs in terms of speedup (5.2× to 8.9× with CAX, but only 3× to 5× with MDMX over the baseline

performance) on the same representative SIMD array architecture. CAX also outperforms MDMX on both area efficiency (a 75% increase versus a 25% increase) and energy efficiency (a 75% increase versus a 24% increase), resulting in better component utilization and sustainable battery life. Furthermore, CAX improves the performance and efficiency with a mere 3% increase in the system area and a 5% increase in the system power, while MDMX requires a 14% increase in the system area and a 16% increase in the system power.

The rest of this chapter is organized as follows. Section 4.2 discusses related research. Section 4.3 describes the modeled architectures and a methodology infrastructure for the evaluation of CAX on a specified SIMD array. Section 4.4 evaluates the system area and power of the modeled architectures, and Section 4.5 analyzes execution performance and efficiency for each case. Section 4.6 concludes this chapter.

4.2 Related Research

Research dealing with harnessing the data-level parallelism (DLP) inherent in color image and video processing applications can be divided into two different groups: (1) those evaluating the performance of current multimedia extensions [6][71][51] and (2) those evaluating the performance of highly parallel architectures [34][88][48]. Numerous research groups and individuals have addressed the effectiveness of multimedia extensions (e.g., Intel MMX, Sun VIS, and MIPS MDMX) for multimedia applications on general-purpose processors. Ranganathan *et al.* [71] analyzed the performance of image and video processing applications on an UltraSPARC processor with and without the VIS media extensions. They observed that a four-way issue, out-of-

order processor provided 2.3× to 4.2× performance improvement over a single-issue, in-order processor, and the VIS extensions provided an additional 1.1× to 4.2× performance improvement. Bhargava *et al.* evaluated the MMX extensions for a set of DSP and multimedia applications on the x86 architectures [6]. In their study, the image applications were the best suited for MMX because the images were stored in a large array of eight-bit data and properly aligned on eight-byte boundaries, showing an average speedup 5.5× and an 81% reduction in dynamic instruction count.

Different subword parallelism alternatives (e.g., matrix-oriented multimedia ISA called MOM and complex streaming instructions called CSI) for multimedia processing applications have been evaluated in [18][42]. Unlike commercial multimedia extensions that are restricted to a single row, both MOM and CSI support two-dimensional data streams, achieving an average of 20% performance gain over the MMX and MDMX extensions with respect to multimedia applications. Overall, existing multimedia-based extensions in general-purpose processors provide moderate performance improvement (2× to 6× speedup) by exploiting subword parallelism. However, their performance is limited in dealing with both color data that are not aligned on boundaries that are a power of two and storage data types that are inappropriate for computation. Moreover, general-purpose processors enhanced with multimedia extensions will not meet the much higher levels of performance required by emerging multimedia applications since they lack the ability to exploit the full data parallelism available in these applications.

SIMD array architectures are geared for data parallelism-rich media applications because they can efficiently exploit massive amounts of data parallelism without complicated control flow or an excessive amount of inter-processor communication.

Massively data parallel arrays of processors have been applied to image processing for almost three decades, but early SIMD machines (e.g., the TMC Connection Machine 1 [83]) were limited by I/O technology. Later machines (e.g., TMC CM-2 [81] and MarPar MP-2 [57]) overcame these limitations through the use of large parallel disk arrays to buffer images. However, these systems achieved generality by sacrificing low cost and portability. Although the fine-grain parallel processing architectures MGAP [40] and ABACUS [7] addressed portability issues, performance was affected by their limited I/O bandwidth and reconfiguration latency, resulting in low resource utilization.

Unlike these SIMD machines, our baseline architecture, the SIMPil array, benefits from directly coupling sensors and processors, alleviating I/O bandwidth bottlenecks, and from short wire lengths, providing compact area and energy efficiency for portable multimedia systems [13][34][8]. While such 2-D SIMD arrays exploit massive data parallelism inherent in 2-D image sequences by operating the same instruction sequences simultaneously on a large number of discrete data sets, their performance is limited by the vector processing of 3-D color data performed within innermost loops. This chapter provides a new solution to support color imaging applications by combining the properties of the human perceptual color space (e.g., YCbCr), color subword parallelism, and SIMD array architecture.

4.3 Methodology

This section describes the modeled architectures and a methodology infrastructure for the evaluation of the CAX instruction set on a representative SIMD array architecture.

4.3.1 Modeled Embedded SIMD Architectures

The SIMD Pixel (SIMPil) processor [13][34] is used as the baseline SIMD imaging architecture for this study. Figure 38 shows the microarchitecture of the SIMD array system, along with its interconnection network. When data are distributed, the processing elements (PEs) execute a set of instructions in a lockstep fashion. With 4×4 pixel sensor sub-arrays, each PE is associated with a specific portion (4×4 pixels or 16 pixel-per-processing-element) of an image frame, allowing streaming pixel data to be retrieved and processed locally. Each PE has a reduced instruction set computer (RISC) datapath with the following minimum characteristics:

- Small amount of local storage (128 32-bit words),
- Three-ported general-purpose registers (16 32-bit words),
- ALU – computes basic arithmetic and logic operations,
- Barrel shifter – performs multi-bit logic/arithmetic shift operations,
- MACC – multiplies 32-bit values and accumulates into a 64-bit accumulator,
- Sleep – activates or deactivates a PE based on local information,
- Pixel unit – samples pixel data from the local image sensor array,
- RGB2YCC and YCC2RGB unit– converts RGB to/from YCbCr,
- ADC unit – converts light intensities into digital values, and
- Nearest neighbor communications through a NEWS (north-east-west-south) network and serial I/O unit.

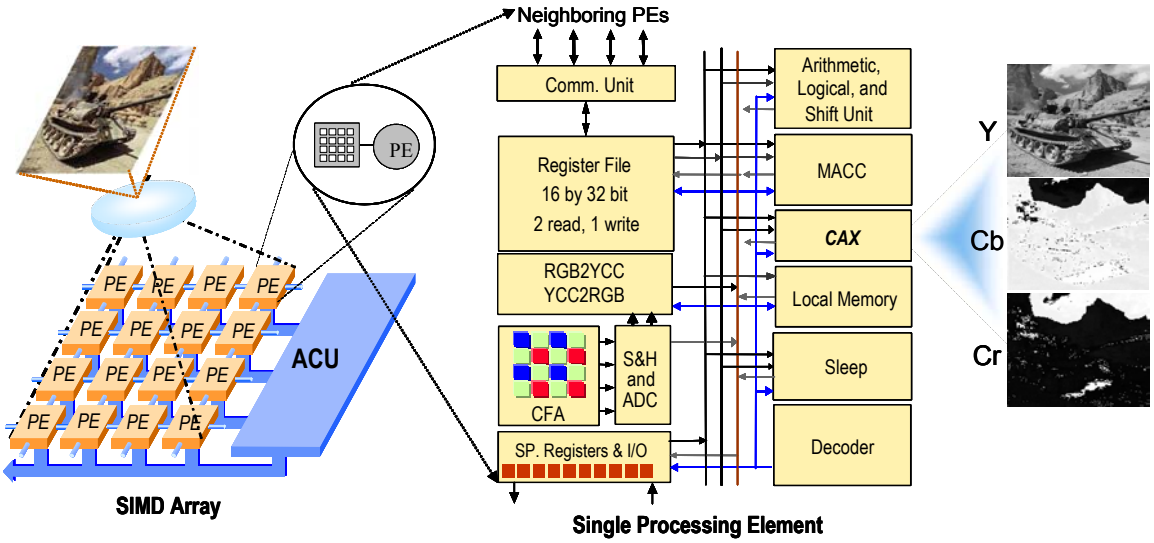


Figure 38. Block diagram of a SIMD array and a processing element.

Despite high performance and energy efficiency from short wire lengths and a specialized microarchitecture, such SIMD imaging systems are not amenable to the vector processing of 3-D color data. In particular, since the 3-D vector computation is performed within innermost loops, its performance does not scale with larger PE arrays. To overcome this performance limitation, the CAX instructions are included in the ISA of the SIMPil array. For a fair performance comparison, we also add MDMX-type instructions to the SIMPil ISA, including additional instructions, such as absolute-differences-accumulation or parallel-conditional-move, equivalent to the CAX instructions. Thus, MDMX (containing 30 instructions) and CAX (containing 34 instructions) have similar instructions, except for the permute instructions. In the experiment, the overhead of the color conversion was not included in the performance evaluation for all the versions. In other words, this study assumes that the baseline, MDMX, and CAX versions directly support YCbCr data in the band-interleaved format (e.g., |Unused|Cr|Cb|Y| for baseline and MDMX and |Cr|Cb|Y|Cr|Cb|Y| for CAX).

Moreover, since the CAX version requires smaller pixel word storage for color imaging applications than the baseline and MDMX versions, this study assumes that the CAX version uses a 64 32-bit word memory, but both baseline and MDMX versions require a 128 32-bit word memory. These memory sizes are sufficient to complete the selected application suite. Table 8 summarizes the parameters of the modeled architectures. An overall simulation infrastructure is presented next.

Table 8. Modeled architecture parameters.

Parameter	Value
System Size	44×38 (1,584 PEs)
Image Sensor per PE (pixel per PE ratio)	4×4 (16 PPE)
VLSI Technology	100 nm
Clock Frequency	80 MHz
Interconnection Network	Mesh
intALU/intMUL/Barrel Shifter/intMACC/Comm	1 / 1 / 1 / 1 / 1
MDMX/CAX: intALU/intMACC	1 / 1
Local Memory Size (baseline/MDMX/CAX)	128 word / 128 word / 64 word

4.3.2 Methodology Infrastructure

Figure 39 shows a methodology infrastructure for this study that is divided into three levels: application, architecture, and technology. At the application level, a set of color imaging applications (e.g., chroma-keying, color edge detection, the scalar median filter, the vector median filter, and vector quantization, and motion estimation) is written in the SIMD assembly language and executed through an instruction-level SIMD simulator, called SIMPilSim. SIMPilSim, shown in Figure 40, allows profiling execution statistics such as cycle count, dynamic instruction histogram, PE utilization, and PE memory usage for the three different versions of the programs: (1) baseline ISA without

sumbword parallelism (SIMPil), (2) baseline plus MDMX ISA (MDMX-SIMPil), and (3) baseline plus CAX ISA (CAX-SIMPil).

At the architecture level, the heterogeneous architectural modeling (HAM) of functional units for SIMD arrays proposed by Chai *et al.* [14][15] is used to calculate the design parameters of the modeled architectures. For the design parameters of the MDMX and CAX functional units (FUs), Verilog models for the baseline, MDMX, and CAX FUs were implemented and synthesized with the Synopsys design compiler (DC) using a 0.18-micron standard cell library. The reported area specifications of the MDMX and CAX FUs were then normalized to the baseline FU, and the normalized numbers were applied to the HAM tool for determining the design parameters of MDMX- and CAX-SIMPil. The design parameters are then passed to the technology level.

At the technology level, the Generic System Simulator (GENESYS) developed at Georgia Tech [65][28] is used to calculate technology parameters (e.g., latency, area, power, and clock frequency) for each configuration. Finally, the database (e.g., cycle times, instruction latencies, instruction counts, area, and power of the functional units), obtained from the application, architecture, and technology levels, is combined to determine execution times, area efficiency, and energy efficiency for each case. The next section presents the system area and power of the modeled architectures.

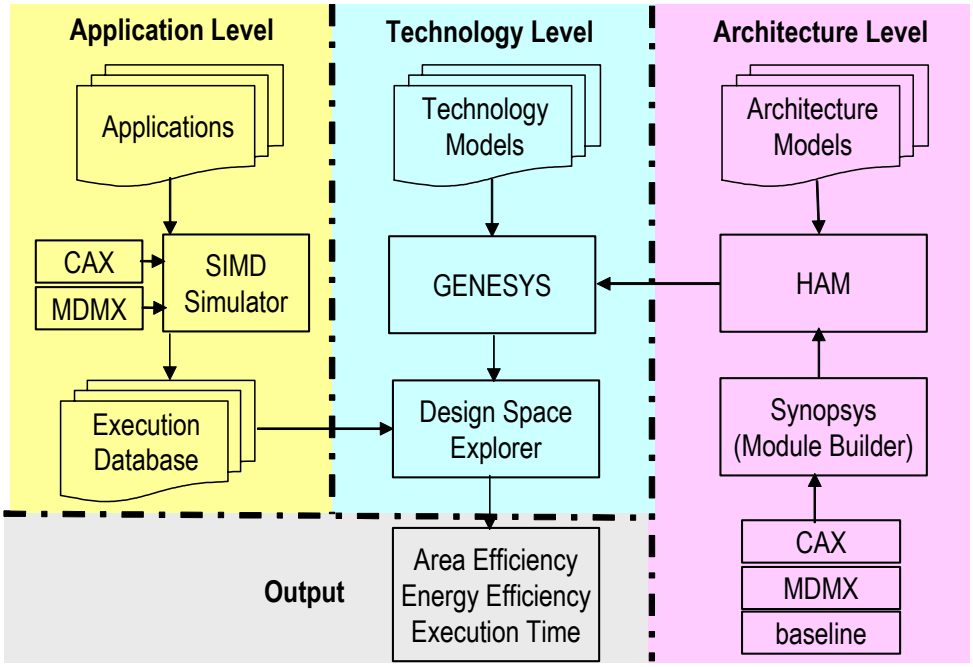


Figure 39. A methodology framework for exploring the design space of three modeled architectures: baseline SIMPil, MDMX-SIMPil, and CAX-SIMPil.

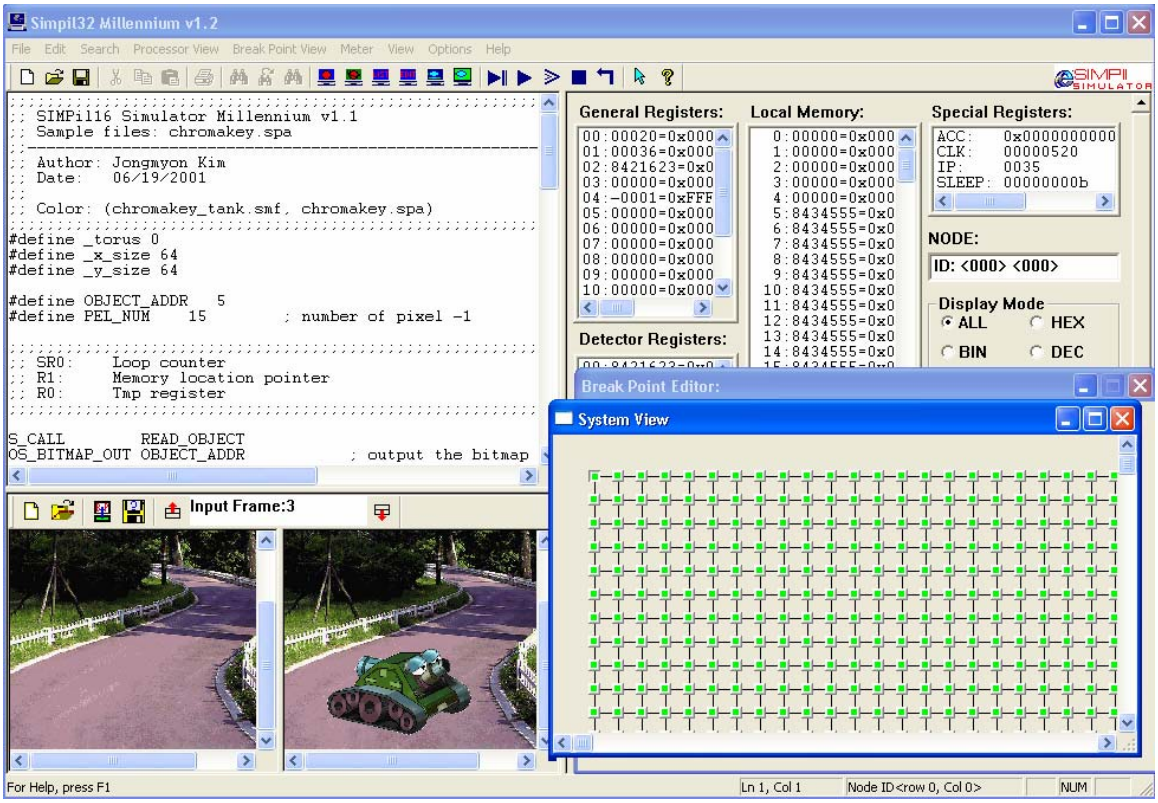


Figure 40. A screenshot of the SIMPil simulator during the chroma-keying process.

4.4 System Area and Power Evaluation using Technology Modeling

GENESYS, an analytical technology modeling tool with macro cell capability, is used to evaluate the system area and power of three modeled architectures: (1) baseline SIMPil, (2) MDMX-SIMPil, and (3) CAX-SIMPil. GENESYS, introduced in [59], integrates a hierarchical set of models that capture key limits such as fundamental, material, device, circuit, and system, shown in Figure 41. The first three levels capture the physical effects of material properties and switching device behaviors. The circuit level estimates all components of the signal propagation delay through a gate. The system level contains architecture, interconnect, and packaging details of a single chip. GENESYS has been calibrated using the Semiconductor Industry Association's International Technology Roadmap for Semiconductors (ITRS) predictions [74] and data from a wide range of implemented ASICs. Complete details on GENESYS and its constituent models can be found in [28][65].

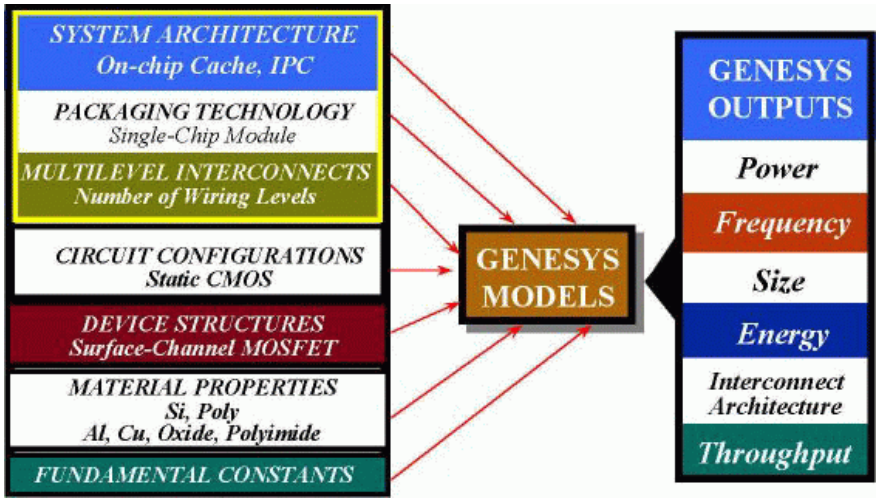


Figure 41. GENESYS system hierarchy.

When design parameters (e.g., gate count, gate depth, Rents' parameters, and average activity factor) from an architectural block (macro cell) are given as input,

GENSYS calculates the functional performance of each unit and the entire processor model such as processor area, cycle time, wire delay, dynamic energy, and static power for a specified technology. Architectural studies of diverse systems, including SIMD arrays [14] and multiprocessor clusters [20], have used GENESYS for the design exploration of the systems. To build the design specifications of the three modeled architectures, the HAM of functional units for SIMD arrays [14] and the Synopsys design compiler are used. GENESYS then combines the design parameters of each architecture configuration while calculating the system area and power of each functional unit and the entire architecture. Table 9 shows system area and power estimates for the modeled architectures.

Table 9. Area and power estimates for three different SIMPil architectures running at 80MHz.

	Baseline SIMPil	MDMX-SIMPil	CAX-SIMPil
Estimated Peak Power [W]	2.7	3.1	2.8
System Area [mm^2]	114	129	117

Figure 42 presents additional data showing the system area and power of MDMX-SIMPil and CAX-SIMPil, normalized to the baseline SIMPil. Experimental results indicate that MDMX requires a 14% increase in the entire system area and a 16% increase in the system power. However, CAX only requires a 3% increase in the system area and a 5% increase in the system power because of the reduced pixel word storage (local memory). These system area and power results are combined with application simulations (e.g., issued instructions and cycle times) for determining execution time, area efficiency, and energy efficiency for each case, which is presented next.

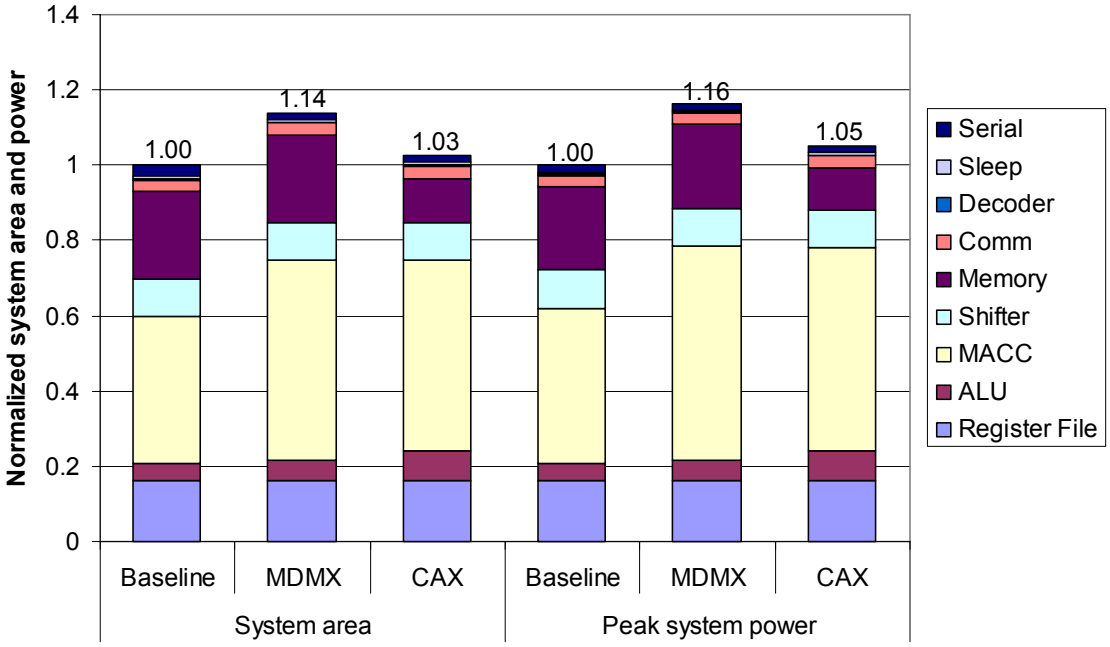


Figure 42. System area and power of MDMX-SIMPil and CAX-SIMPil, normalized to the baseline SIMPil.

4.5 Experimental Results

Cycle accurate simulation and technology modeling are used to determine the performance and efficiency characteristics of the modeled architectures for each application task. The three versions of the programs (e.g., baseline ISA, baseline plus MDMX ISA, and baseline plus CAX ISA) are developed in their respective assembly languages for the SIMPil system, in which all three versions for each program have the same parameters, data sets, and calling sequences. Their execution statistics (e.g., instruction count, execution cycle count, and PE utilization) are then combined with GENESYS predictions to evaluate each benchmark's energy consumption, energy efficiency, and area efficiency. The metrics of the execution cycle count, corresponding energy consumption, sustained throughput, energy efficiency, and area efficiency of each case form the basis of the study comparison, defined in Table 10.

Table 10. Summary of evaluation metrics.

execution time	sustained throughput	energy efficiency	area efficiency
$t_{exec} = \frac{C}{f_{ck}}$	$Th_{sust} = \frac{O_{exec} \cdot U \cdot N_{PE}}{t_{exec}}$	$\eta_E = \frac{O_{exec} \cdot U \cdot N_{PE}}{Energy} [\frac{\text{Gops}}{\text{Joule}}]$	$\eta_A = \frac{Th_{sust}}{Area} [\frac{\text{Gops}}{\text{s} \cdot \text{mm}^2}]$

C is the cycle count, f_{ck} is the clock frequency, O_{exec} is the number of executed operations, and N_{PE} is the number of processing elements. Note that since each CAX and MDMX instruction executes more operations (typically six and three times, respectively) than a baseline instruction, we assume that each CAX, MDMX, and baseline instruction executes six, three, and one operation, respectively, for the sustained throughput calculation.

4.5.1 Execution Performance Evaluation Results

This section discusses the impact of CAX on execution performance for the selected color imaging applications on the SIMPil system. This section also presents detailed application implementations with CAX for further insights into the CAX behavior.

4.5.1.1 Overall Results

Figure 43 illustrates execution performance (speedup in executed cycles) attained by CAX and MDMX when compared with the baseline performance without subword parallelism. The results indicate that CAX outperforms MDMX for all the programs in terms of speedup, indicating a speedup ranging from 5.2× to 8.9× (an average of 6.7×) with CAX, but only 3× to 5× (an average of 3.8×) with MDMX over the baseline performance.

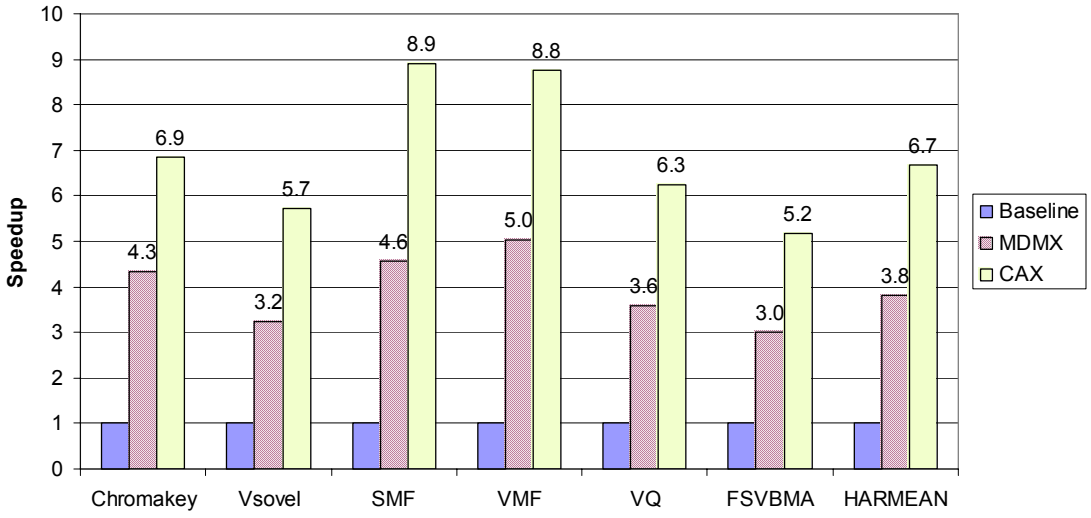


Figure 43. Speedups of the CAX and MDMX versions over the baseline performance.

CAX also achieves higher sustained throughput (an average of 194 Gops/sec) than the MDMX version (an average of 155 Gops/sec) and the baseline version (an average of 113 Gops/sec) across all the application tasks. Table 11 summarizes the execution parameters for each case in the SIMPil system. Note that the scalar execution time was not included in the sustained throughput because vector instructions dominate instruction histograms in which each issued vector instruction is multiplied by the number of active processing elements (1,584 PEs in this study). In other words, scalar instructions execute in the array controller unit concurrently with vector instructions executing in PEs, and the scalar execution time is effectively hidden during vector execution [34]. The next section presents an in-depth analysis of the CAX effectiveness for each benchmark program.

Table 11. Application performance of the baseline, MDMX, and CAX versions on a 1,584 PE system running at 80 MHz.

Application	ISA	Memory Size		Vector Instruction	Scalar Instruction	System Utilization [%]	Sustained Throughput [Gops/sec]
		PE [Byte]	System [KB]				
Chromakey	Baseline	192	768	1,227	106	88	112
	MDMX	192	768	283	106	100	155
	CAX	128	512	179	58	100	183
VSobel	Baseline	344	1,376	7,177	1,011	100	122
	MDMX	344	1,376	2,117	371	100	160
	CAX	216	864	1,257	195	100	207
SMF	Baseline	244	976	43,287	2,771	71	89
	MDMX	244	976	9,495	2,771	100	181
	CAX	172	688	4,863	1,395	100	260
VMF	Baseline	244	976	37,397	7,891	93	118
	MDMX	244	976	7,430	3,027	97	158
	CAX	172	688	4,264	1,523	94	203
VQ	Baseline	204	816	180,551	20,755	91	115
	MDMX	204	816	50,503	20,755	97	133
	CAX	136	544	28,863	11,715	94	151
FSVBMA	Baseline	196	784	97,229	10,043	95	120
	MDMX	196	784	32,502	10,379	98	140
	CAX	100	400	18,784	7,706	96	159

4.5.1.2 Benefits of CAX for Color Imaging Applications

Figure 44 shows the distribution of issued vector instructions for the SIMPil system with MDMX and CAX, normalized to the baseline version. Each bar divides the instructions into the arithmetic-logic-unit (ALU), memory (MEM), communication (COMM), PE activity control unit (MASK), image pixel loading (PIXEL), MDMX, and CAX. The use of CAX reduces a significant number of the instruction counts for all of the programs. In particular, CAX reduces a significant number of ALU and memory instruction counts due to its instruction definition. An interesting observation is that unlike the results for the superscalar processor (see Figure 35), the FSVBMA program has the smallest reduction in the instruction count with CAX. This is because it involves high inter-PE computation operations that are not affected by CAX. For example, each PE cannot directly support a macroblock size of 16×16 pixels because 4×4 pixels are

mapped to each PE. As a result, the 4×4 distortions are computed in each PE separately. Each result is then combined through NEWS communication instructions for the final distortion between the 16×16 input and reference blocks.

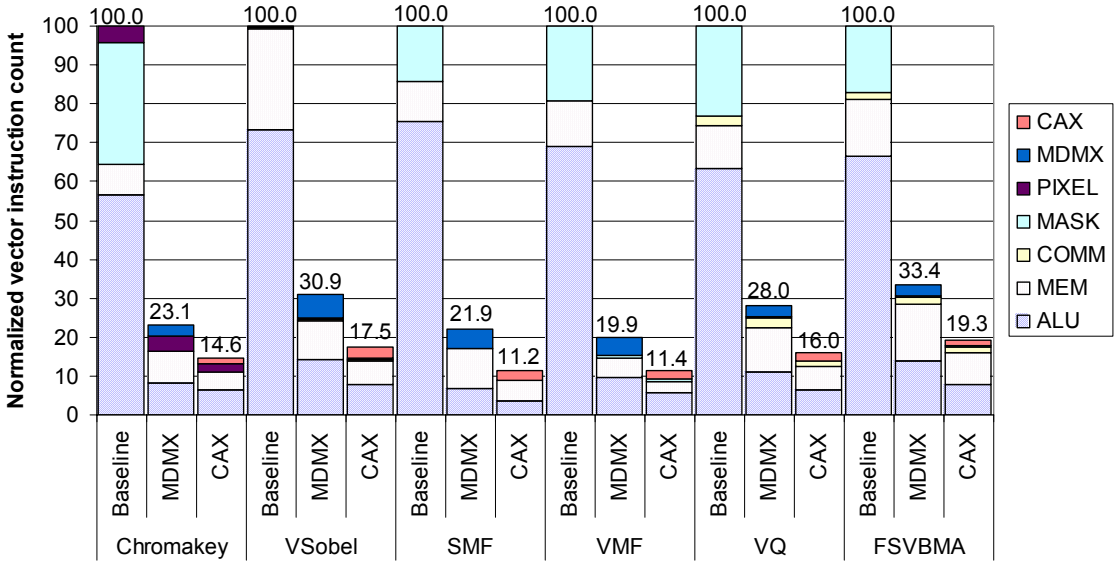
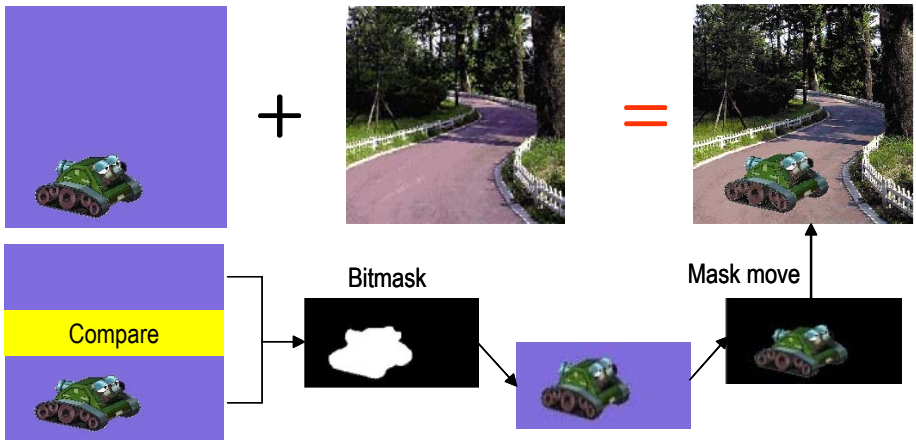


Figure 44. The distribution of issued vector instructions for the SIMPil system with CAX and MDMX, normalized to the baseline version.

Chroma-keying. Chroma-keying is an image overlay technique that is used extensively to produce special effects (e.g., on television weather programs, the image of the weather person is overlaid on the weather map image). This application demonstrates how conditional selection using CAX removes branch mispredictions (or MASK instructions) while performing multiple selection operations in parallel. In the study, the chroma-keying program is performed in the YCbCr color space. Figure 45 shows the required C code operation and the corresponding CAX assembly code implemented on SIMPil, along with comments and a pictorial representation.



(a)

```
for (i = 0; i < # of pixels; i++) {
    if (x[i] == blue backing) new_image[i] = y[i];
    else new_image[i] = x[i]; // foreground
}
```

(b)

```
Loop
pload r1, mem1 ; load two-packed YCbCrs from the picture with the tank object
pload r2, mem2 ; load two-apcked YCbCrs from the blue background frame
pload r3, mem2 ; load two-packed YCbCrs from the picture with the pine road
cmpeqn_crcby r4, r1, r2; if r1(i) != r2(i), r4(i) is all ones. Otherwise, all zeros.
cmov_crcby   r3,r2,r4; if r4(i) == all ones,  r3(i) ← r2(i). Otherwise, no ops.
```

(c)

Figure 45. The procedure of a chroma-keying application: (a) a pictorial representation, (b) required C code, and (c) CAX assembly code. Note that the MDMX assembly code has the same functional instructions for CAX except that it loads and processes a packed YCbCr in a 32-bit register.

The `PLOAD` instruction loads two-packed YCbCr data from the three pictures: (1) tank on a blue background, (2) blue background, and (3) pine road. The `CMPEQN_CRCBY` instruction then compares pixels from the tank frame and the equivalent pixels from the blue background while building a mask that is a sequence of all ones or all zeros in the Y, Cb, and Cr operands of the register. This mask is used on the same two-packed YCbCr data from the tank frame and the equivalent two-packed YCbCr data from the pine road. The conditional move instruction `CMOV_CRCBY` uses the mask in order to overlay pixels from the tank object onto the pine road. These CAX instructions remove many MASK instructions while reducing a large number of ALU instructions by processing multiple selection operations in parallel. Table 12 illustrates a

comparison of instruction counts using the baseline, MDMX, and CAX ISAs for a conditional selection operation of 4×4 pixels. The instruction count decreases 82% with CAX, but only 71% with MDMX over the baseline version.

Table 12. A comparison of instruction counts using the baseline, MDMX, and CAX ISAs for a conditional selection operation of 4×4 pixels.

	Baseline	MDMX	CAX
ALU	693	101	77
MEM	99	99	59
MASK	384	-	-
PIXEL	51	51	27
MDMX	-	32	-
CAX	-	-	16
Scalar Instructions	106	106	58
Total	1,333	389	237

Color Edge Detection. Edge detection is a fundamental task in image processing. Many image applications, such as object recognition and image segmentation, depend on the accuracy of edge detection. Unlike monochrome edge detection that may not be sufficient in color images when neighboring objects have different hues but equal intensities, color edge detection accounts for local changes in both luminance and chrominance components to provide crucial information conveyed by color. In this study, color edge detection based on a Sobel operator is implemented on the SIMPil simulator. Figure 46 illustrates the procedure of the color edge detection implementation using CAX. Each of two-packed 16-bit YCbCr data is loaded into registers, and some pixels are rearranged with the `ROTATE_CRCBY` and `MIX_CRCBY` instructions for an efficient format of the multiply-accumulate computation, which involves a vector pixel and its eight neighbors within a 3×3 window. Also, each of the coefficient values saved in memory is loaded and then distributed into the Y, Cb, and Cr positions of the target register with the

BCAST_CRCBY instruction. The MACC_CRCBY instruction then multiplies pairs of sub-elements in the two source registers (one for color components and the other for coefficients) while accumulating each result in the packed accumulator. Furthermore, two window boxes are efficiently processed in parallel. This implementation using CAX leads to a large reduction in the instruction count while reducing register pressure and memory traffic. Table 13 presents a comparison of instruction counts using the baseline, MDMX, and CAX ISAs for a Sobel operation of two 3×3 window pixels. The instruction count decreases 84% with CAX, but only 68% with MDMX over the baseline version.

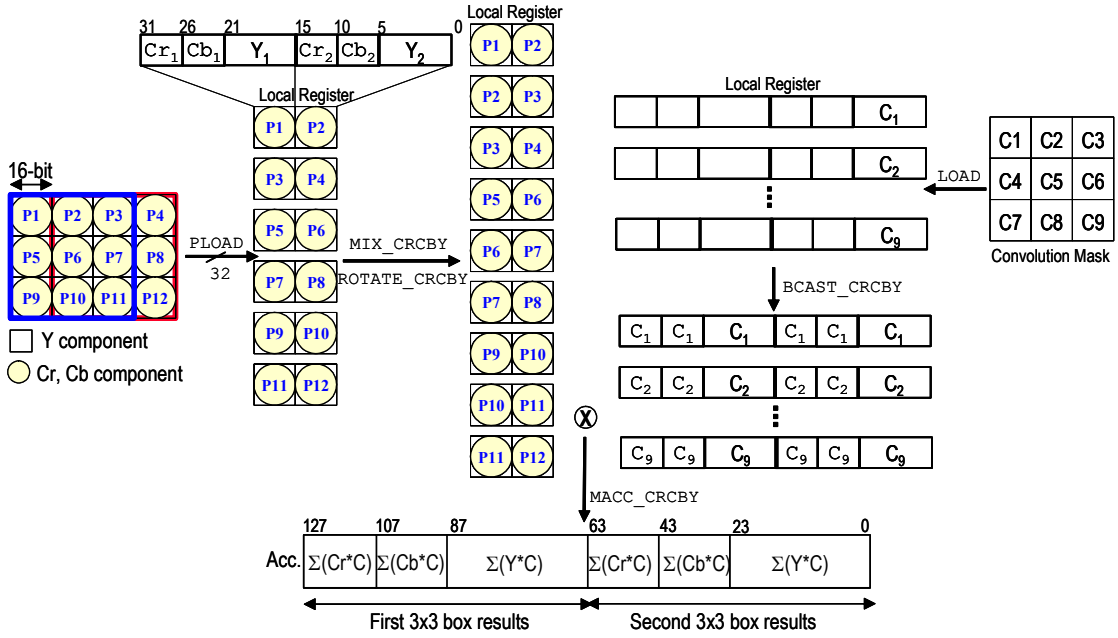


Figure 46. The procedure of a color edge detection implementation using the CAX instructions.

Table 13. A comparison of instruction counts using the baseline, MDMX, and CAX ISAs for a Sobel operation of two 3×3 window pixels.

	Baseline	MDMX	CAX
ALU	344	82	43
MEM	110	38	20
MDMX	-	22	-
CAX	-	-	11
Scalar Instructions	54	22	10
Total	508	164	84

The Vector Median Filter. The well-known vector median filter (VMF) is widely used to filter out noise from color images [69]. In this study, the VMF using the YCbCr channels (see Section 2.4.1) is implemented on the SIMPil system. The most time-critical operation for this implementation is the sum of pixel differences between pixels in the window of size N×N (3×3 in this study). With the ADACC_CRCBY instruction, a VMF operation can be performed on the two window blocks of pixels in parallel. In particular, ADACC_CRCBY calculates the absolute differences of pairs of the sub-elements in the two source registers while accumulating each result in the packed accumulator. Thus, one ADACC_CRCBY instruction reduces several baseline ALU operations and memory accesses for intermediate results (since immediate results are stored in the accumulator rather than in memory). Table 14 presents the number of instruction counts using the baseline, MDMX, and CAX ISAs for computing the median within two 3×3 window pixels. The instruction count decreases 89% with CAX, but only 79% with MDMX over the baseline version.

Table 14. A comparison of the number of instructions using the baseline, MDMX, and CAX ISAs for computing the median within two 3×3 window pixels.

	Baseline	MDMX	CAX
ALU	3,510	414	248
MEM	499	178	98
MASK	448	16	16
MDMX	-	160	-
CAX	-	-	80
Scalar Instructions	948	338	169
Total	5,355	1,106	611

The Scalar Median Filter. Like the VMF, the scalar median filter (SMF) is also a noise reduction technique that eliminates impulse noise spikes from an image by taking the median pixel value in a 3×3 window that is stepped across the entire image. However, the SMF differs from the VMF in that it separately replaces each corrupted color component (e.g., Y, Cb, and Cr) with the median from the reference and its neighboring components.

The most computationally intensive operation of the SMF implementation is to find the median pixel value from the nine pixels in the processing window. The `MIN_CRCBY` and `MAX_CRCBY` instructions accelerates the bubble sorting algorithm by comparing pairs of sub-elements in the two source registers while outputting the minimum or maximum values of the corresponding sub-elements. These instructions lead to a significant reduction in the ALU and MASK instruction counts. Table 15 presents a comparison of instruction counts using baseline, MDMX, and CAX ISAs for a sorting operation of two 3×3 widow pixels. The instruction count decreases 88% with CAX, but only 75% with MDMX over the baseline version.

Table 15. A comparison of instruction counts using the baseline, MDMX, and CAX ISAs for a sorting operation of two 3×3 window pixels.

	Baseline	MDMX	CAX
ALU	4,374	312	157
MEM	516	516	259
MASK	384	-	-
MDMX	-	256	-
CAX	-	-	128
Scalar Instructions	308	308	154
Total	5,582	1,392	698

Vector Quantization. Full search vector quantization (VQ) [35] is an attractive technique for low rate and low power image and video compression. It has a computationally inexpensive decoding process and low hardware requirement for decompression, while still achieving an acceptable picture quality at high compression ratios. However, the encoding process is computationally very intensive. Computational cost can be reduced by using suboptimal approaches such as tree-searched vector quantization (TSVQ) [35]. In this study, a parallel implementation of full search VQ is implemented on a SIMD array system to overcome this computational burden. VQ is defined as a mapping of k-dimensional vectors in vector space $\mathbf{R}^k$ onto a finite set of vectors $\boldsymbol{V} = \{ \boldsymbol{y}_i \; ; \; i = 1,\ldots,N\}$, where N is the size of the codebook. Each vector $\mathbf{y}_i = (y_0,\ldots,y_{k-1})$ is called a codebook vector or codeword. Only index i of the resulting code vector is sent to the decoder. At the decoder, an identical copy of the codebook is retrieved as the encoder by a simple table-lookup operation. The compression ratio depends on the cardinality of the codebook, usually much smaller than that of the input domain.

In this implementation, a codebook of 256 4×4 code vectors designed off-line through a standard Linda-Buzo-Gray (LBG) training process is used to achieve a 0.5 bit

per pixel encoding for an image in 24-bit color, using 4×4 (k = 16). In the 2-D case, non-overlapping vectors are extracted from the input image by grouping a number of contiguous pixels to retain available spatial correlation of data. The input blocks are then compared with the codebook in a parallel systolic fashion, with a large number of them compared at any given time in parallel. A key enabling role is played by the toroidal structure of the interconnection network, which enables communication among the nodes on opposite edges of the mesh.

The most time-critical operation for this implementation is the distortion calculation between a 4×4 input block and a local codeword. The distortion can be efficiently calculated with the `ADACC_CRCBY` instruction by comparing pairs of sub-elements in the two source registers while accumulating each result in the packed accumulator. Table 16 shows a comparison of instruction counts using the baseline, MDMX, and CAX ISAs for a full search VQ operation of 4×4 pixels. The instruction count decreases 88% with CAX, but only 81% with MDMX over the baseline version.

Table 16. A comparison of instruction counts using the baseline, MDMX, and CAX ISAs for a VQ operation of 4×4 pixels.

	Baseline	MDMX	CAX
ALU	483	37	29
MEM	80	34	18
MASK	34	-	-
MDMX	-	17	-
CAX	-	-	9
Scalar Instructions	34	34	18
Total	631	122	74

Motion Estimation. Motion estimation (ME) is a core building block in several video compression standards (e.g., H.26x and MPEG). Compression is achieved through a block-matching algorithm (BMA) that subdivides the current frame into small reference

blocks and then finds the best match for each block among the available blocks in the previous frame. In this implementation, the macroblock size of 16×16 pixels and the search range of ±8 are used. Since the objective of this study is to achieve accurate motion estimates, both luminance and chrominance components are used in the program (i.e., FSVBMA) rather than only the luminance component in the standard BMA (see Section 2.4.3).

The most time-critical operation is the sum of mean absolute differences (MAD) computation that involves a reference block of pixels and all the candidate blocks of pixels in the search area. Similar to the VQ implementation, the MAD block is efficiently processed with the `ADACC_CRCBY` instruction by comparing pairs of the sub-elements in the two source registers (one containing pixels within the candidate block; the other containing pixels within the reference block) while accumulating each result in the packed accumulator. This process is iterated until all the candidate blocks are compared by the reference block. Table 17 shows a comparison of instruction counts using the baseline, MDMX, and CAX ISAs for a MAD computation of 16×16 pixels. The instruction count decreases 85% with CAX, but only 75% with MDMX over the baseline version.

Table 17. A comparison of the number of instructions using the baseline, MDMX, and CAX ISAs for a MAD operation of 16×16 pixels.

	Baseline	MDMX	CAX
ALU	392	42	28
MEM	33	33	17
MASK	48	-	-
COMM	6	6	6
MDMX	-	16	-
CAX	-	-	8
Scalar Instructions	33	33	17
Total	512	130	76

Overall, CAX clearly outperforms MDMX in consistently reducing the number of instructions required for each application. For portable multimedia systems, battery life performance and system area are as important as processing performance. An evaluation of energy- and area-related performance is presented in the following sections.

4.5.2 Energy Efficiency Results

Figure 47 shows energy consumption for the SIMPil system with MDMX and CAX, normalized to the baseline version. Each bar divides the energy consumption into the functional unit (FU, combines ALUs, Barrel Shifter, and MACC), storage (combines Register file and Memory), and others (combines Comm., Sleep, Serial, and Decoder) categories. The use of CAX significantly reduces energy consumption for all the programs because of a large reduction in the issued instruction count, in which all the implementations have been examined at the same 80 MHz clock frequency and 100nm technology. (This study assumes that unused units dissipate zero power.) CAX reduces energy consumption from 80% (FSVBMA) to 89% (VMF), while MDMX reduces energy consumption from only 60% (FSVBMA) to 79% (VMF) over the baseline version. As expected, the FSVBMA program using CAX shows the lowest reduction rate in the energy consumption metric because of the smallest reduction rate in the instruction count. Since CAX reduces a significant number of ALU and memory instructions, less energy is spent on the ALU and storage units.

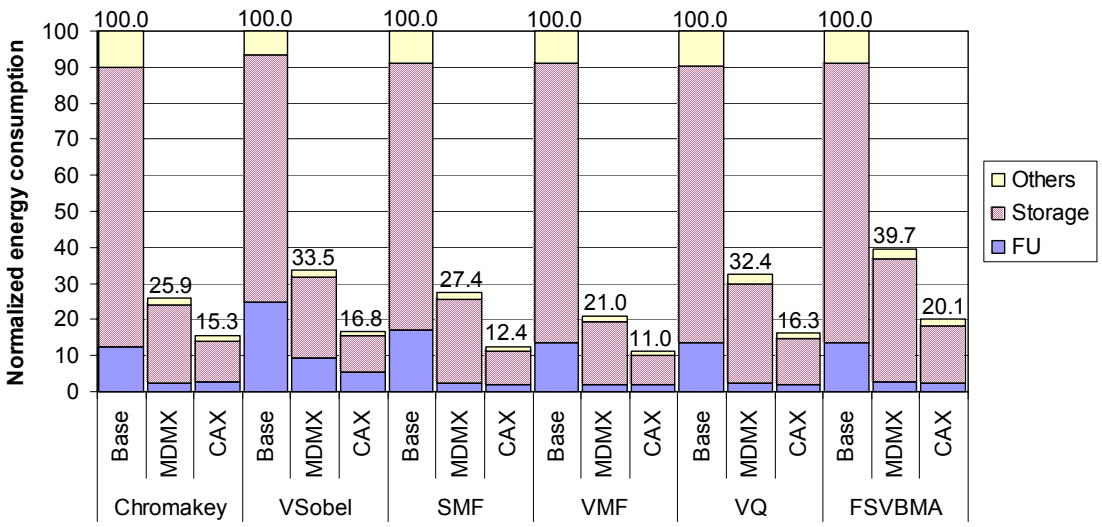


Figure 47. Energy consumption for the SIMPil system with CAX and MDMX, normalized to the baseline version.

Figure 48 presents additional data showing energy efficiency, the task throughput achieved per unit of Joule, for the SIMPil system with MDMX and CAX, normalized to the baseline version. CAX outperforms MDMX across all the programs in the energy efficiency metric, indicating a 65% increase with CAX, but only a 21% increase with MDMX. This is because CAX achieves higher sustained throughputs with a small increase in the system power. Increasing energy efficiency improves sustainable battery life for given system capabilities.

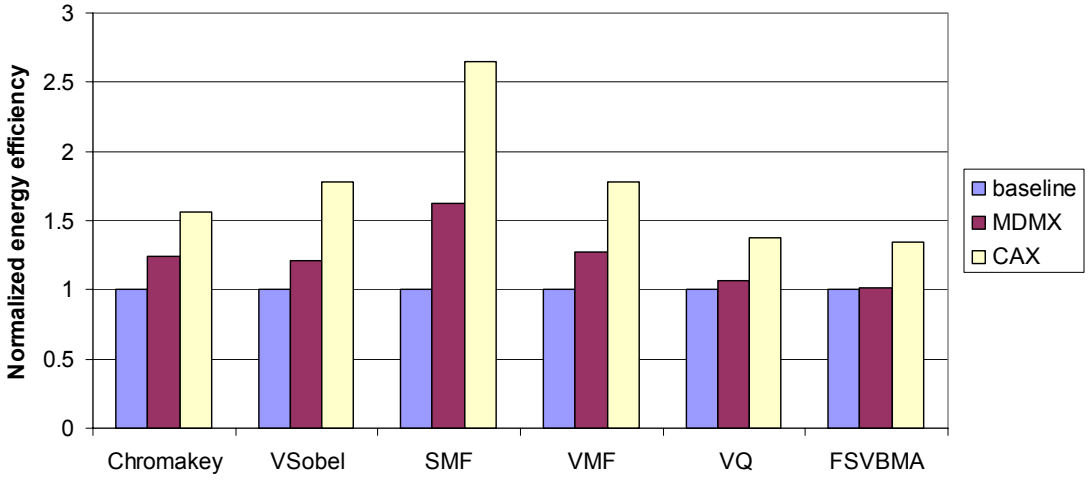


Figure 48. Energy efficiency for the SIMPil system with CAX and MDMX, normalized to the baseline version.

4.5.3 Area Efficiency Results

Area efficiency is the task throughput achieved per unit of area. Figure 49 shows the area efficiency for the SIMPil system with MDMX and CAX, normalized to the baseline version. As with energy efficiency, CAX outperforms MDMX for all the programs in the area efficiency metric, indicating a 66% increase with CAX, but only a 21% increase with MDMX. This is because CAX achieves higher sustained throughput with smaller area overhead. Increasing area efficiency improves component utilization for given system capabilities.

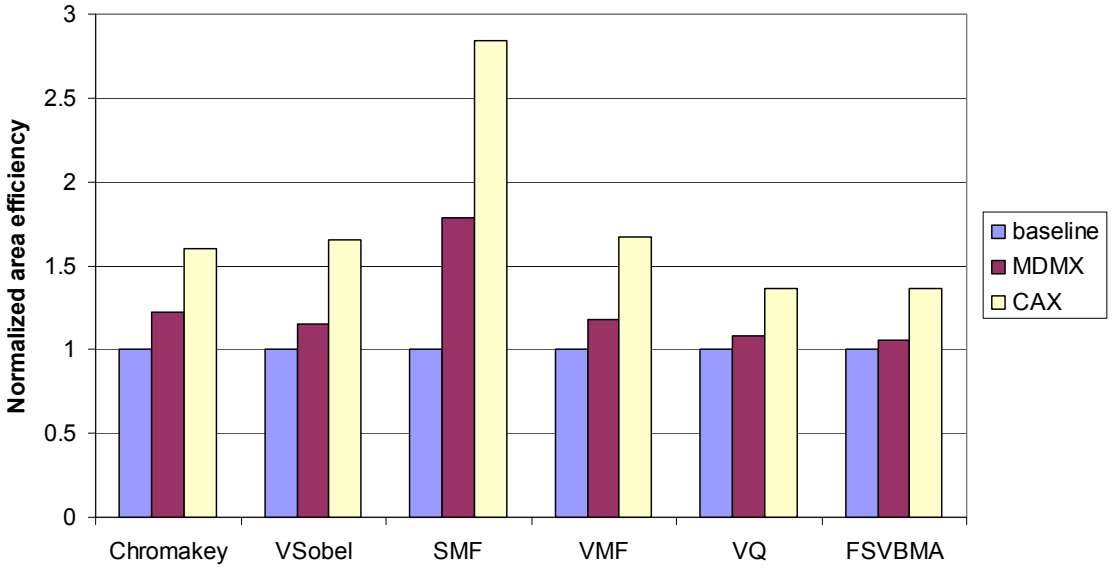


Figure 49. Area efficiency for the SIMPil system with CAX and MDMX, normalized to the baseline version.

4.6 Conclusion

Future embedded imaging products must achieve greater processing performance while maintaining low cost and low energy consumption. Application-specific embedded systems (e.g., 2-D SIMD arrays) have demonstrated the potential to meet the high computational requirements and cost goals. The SIMPil array, for example, benefits from the exploitation of abundant data parallelism inherent in multimedia applications, short wire lengths, and specialized microarchitecture to provide a significant improvement in energy efficiency. While 2-D SIMD arrays, including SIMPil, provide a convenient parallel processing model with moderate generality for processing 2-D image sequences, their performance is limited by the vector processing of 3-D YCbCr channels performed within innermost loops.

The CAX instruction set has been presented to eliminate this performance limitation by including parallel operations on two packed 16-bit YCbCr data into the

instruction set architecture of a representative 32-bit datapath SIMD array. CAX obtains greater concurrency and efficiency for processing color image sequences by harnessing parallelism within the human perceptual color space (e.g., YCbCr) not reachable by other multimedia extensions. In particular, the key findings on a specified SIMD array architecture are the following:

- CAX achieves a speedup ranging from 5.2x to 8.9x (an average of 6.7x) over the baseline performance. This is in contrast to MDMX, which achieves a speedup ranging from 3x to 5x (an average of 3.8x) over the baseline.
- CAX reduces energy consumption from 80% to 89%, while MDMX reduces energy consumption only from 60% to 79% over the baseline version.
- Unlike MDMX, CAX benefits from greater concurrency and reduced pixel word storage. As a result, the area efficiency increases from 36% to 184% (an average of 75%) with CAX, but only 8% to 78% (an average of 25%) with MDMX. In addition, the energy efficiency increases from 35% to 164% (an average of 75%) with CAX, but only 2% to 63% (an average of 25%) with MDMX. Increasing area and energy efficiencies imply augmenting component utilization and sustainable battery life, respectively, for given system capabilities.
- Furthermore, CAX improves the performance and efficiency with a mere 3% increase in the system area and a 5% increase in the system power, while MDMX requires a 14% increase in the system area and a 16% increase in the system power. Although these overheads can be reduced through optimized

design techniques and advanced VLSI technologies, CAX still has the potential to provide higher processing performance and efficiency.

In the next chapter, several CAX-PE architectures based on different vector-pixel-per-processing-element values are analytically studied to identify an ideal design space that delivers sufficient processing performance with the lowest cost and the longest battery life.

CHAPTER 5

ANALYTICALLY DETERMINING OPTIMAL GRAIN SIZES IN EMBEDDED SIMD ARCHITECTURES

5.1 Introduction

A significant issue for focal-plane SIMD image processing architectures is determining the ideal grain size that provides sufficient processing performance with the lowest cost and the longest battery life for target applications. In color imaging applications, the grain size of the processing elements (PEs) determines the number of vector pixels that are mapped to each PE, which is called the vector-pixel-per-processing-element (VPPE) ratio. The VPPE ratio has a significant impact on the overall area and energy efficiency of the computational array.

This chapter explores the effects of different VPPE ratios on performance and efficiency for a specified PE architecture and implementation technology using cycle accurate simulation and analytical technology modeling. Cycle accurate simulation provides execution statistics such as cycle count, dynamic instruction histogram, PE utilization, and PE memory usage. An analytical technology modeling tool estimates technology parameters such as system area, power, latency, and clock frequency. These databases are combined to show the impact of different VPPE values on the performance and efficiency metrics. Moreover, the impact of CAX on each VPPE configuration is evaluated to identify the most efficient PE design that delivers sufficient processing performance with the lowest cost for a specified PE architecture and implementation technology. Experimental results using architectural and workload simulation indicate that CAX outperforms MDMX for all of the VPPE configurations for full search vector

quantization (FSVQ) in terms of processing performance, area efficiency, and energy consumption. Results also suggest that VPPE = 16 with CAX achieves high processing performance with the lowest cost.

The rest of this chapter is organized as follows. The next section discusses related research. Section 5.3 describes the VPPE variation while illustrating the correlation among color image size, VPPE ratio, and PE architecture. Section 5.4 presents modeled SIMD architectures that have different VPPE values and different amounts of local memory. Section 5.5 evaluates system area and power for each VPPE configuration with and without CAX or MDMX using technology modeling. Section 5.6 analyzes execution performance and efficiency for each case. Section 5.7 concludes this chapter.

5.2 Related Research

In the last decade, with the rapid progress in VLSI technology, tremendous numbers of transistors have enabled the monolithic integration of traditional imaging systems such as a charge-coupled device (CCD) array, an analog-to-digital conversion (ADC) unit, and a DSP [30]. The performance of these systems, however, is limited by the serialized communications between the different modules. As a solution, CMOS image sensors allow direct pixel access and enable their ability to be co-located [29] or vertically integrated [72, 8] with the CMOS computing layer. However, none of these systems have addressed the issue of how much processing capability is needed for each PE per pixel directly mapped to it.

Recently, Gentile *et al.* have presented a study to determine the impact of varying granularity of mapping an image to the PE array [33]. In [39], Herbordt *et al.* examined

the effects of varying the array size, the datapath, and the memory hierarchy on both cost and performance. However, these studies measured processing performance and efficiency on sets of grayscale (1-D) image processing applications, failing to provide a quantitative understanding of performance and efficiency with respect to 3-D vector processing for different PE configurations.

This chapter evaluates the effects of different VPPE ratios on performance and efficiency with respect to 3-D image processing for a specified PE architecture and implementation technology. This chapter also evaluates the impact of CAX on each VPPE configuration to identify the most efficient PE granularity.

5.3 Vector-Pixel-per-Processing-Element Ratio

Reconfigurable silicon area usage within an integrated pixel processing array is a key issue for focal-plane SIMD array architectures because of limited chip resources and varying application requirements. To determine the effect of varying silicon area usage on the reference SIMD array, the VPPE ratio (number of vector pixels mapped to each processor within a SIMD architecture) is selected as the design variable in this study. Figure 50 pictorially illustrates the assignment of vector pixels based on the VPPE ratio. In this study, seven VPPE values are used, defined as $\text{VPPE} = 2^{2i}$, $i = 0,\ldots,6$. The corresponding number of processing elements is defined as $N_{PE} = N_{img}/\text{VPPE}$ in which N_{img} is the number of pixels in the image. Since all the configurations use a fixed three-band 256×256 pixel image, the number of PEs in a 256 × 256 pixel system is determined to be $N_{PE} = 2^{2(8-i)}$, $i = 0,\ldots,6$. Different VPPE configurations and their parameters are described next.

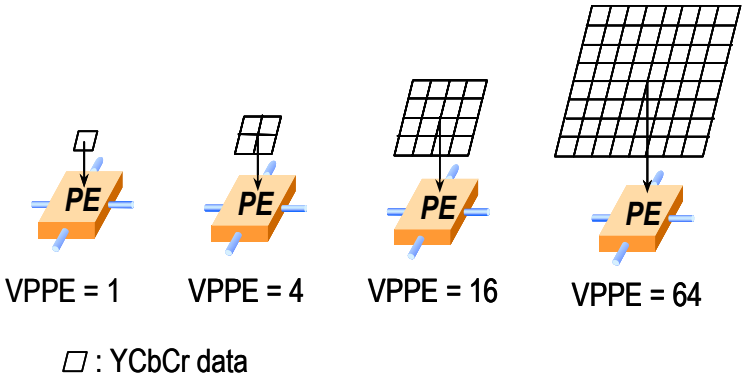


Figure 50. Examples of vector pixels per processing element ratio.

5.4 Modeled PE Architectures

Three different reference architectures (e.g., baseline SIMPil, MDMX-SIMPil, and CAX-SIMPil) are used to evaluate the effects of different VPPE ratios on performance and efficiency. Each configuration has a different VPPE ratio and a different amount of local memory to store input images and temporary data produced during processing. Since each CAX configuration requires smaller pixel word storage than the corresponding baseline and MDMX configurations, the local memory size is set to twice the VPPE ratio for the CAX configurations but four times the VPPE ratio for the baseline and MDMX configurations, except for VPPE = 1 where eight words are used for all three versions. Table 18 describes all the configurations and their local memory sizes. The next section describes the system area and power for each configuration using technology modeling.

Table 18. VPPE configurations and their parameters.

Parameter	Value						
# of PEs	65,536	16,384	4,096	1,024	256	64	16
VPPE values	1	4	16	64	256	1,024	4,096
Base (Memory/PE) [word]	8	16	64	256	1,024	4,096	16,384
MDMX (Memory/PE) [word]	8	16	64	256	1,024	4,096	16,384
CAX (Memory/PE) [word]	8	8	32	128	512	2,048	8,192
VLSI Technology	100 nm						
Clock Frequency	50 MHz						
Interconnection Network	Mesh						
intALU/intMUL/Barrel Shifter/intMACC/Comm	1 / 1 / 1 / 1 / 1						

5.5 System Area and Power Evaluation using Technology Modeling

The GENESYS tool [28] is used to determine implementation characteristics (e.g., system area and power) for each PE configuration. Figures 51 and 52 show system area and power estimations versus VPPE values, respectively, in which all the configurations were examined in the same 100nm technology and 50 MHz node frequency. For VPPEs at or above 256, both system area and power asymptotically approach a lower limit where local memory area dominates. Below this point, however, both system area and power decrease linearly. As a result, a number of configurations are not feasible, requiring silicon area greater than 1,000 mm^2 (the ITRS projected limit in 100 nm CMOS technology). Although some configurations with power above three watts are not feasible as well in terms of battery operation and heat removal, power reduction techniques [49][5][17] (e.g., clock frequency scaling) allow the power dissipation levels required by portable, battery-operated devices at the expense of performance (execution time).

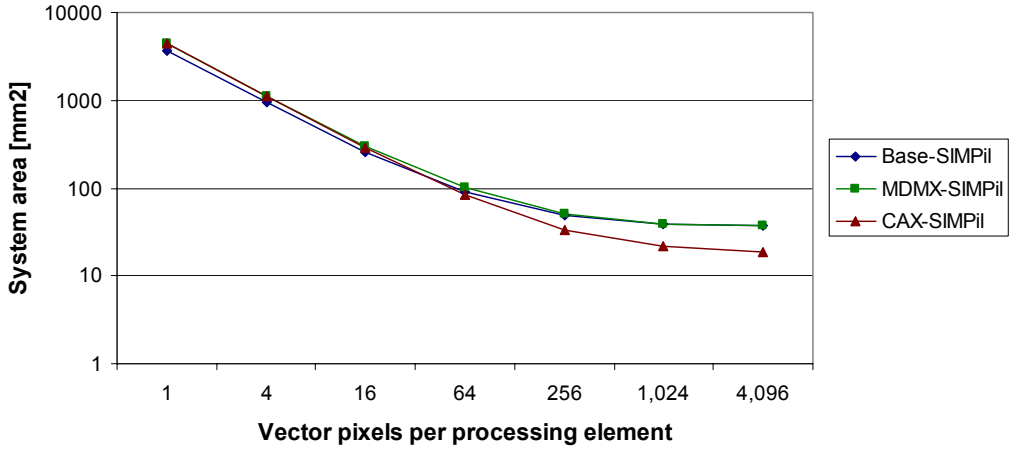


Figure 51. System area versus VPPE.

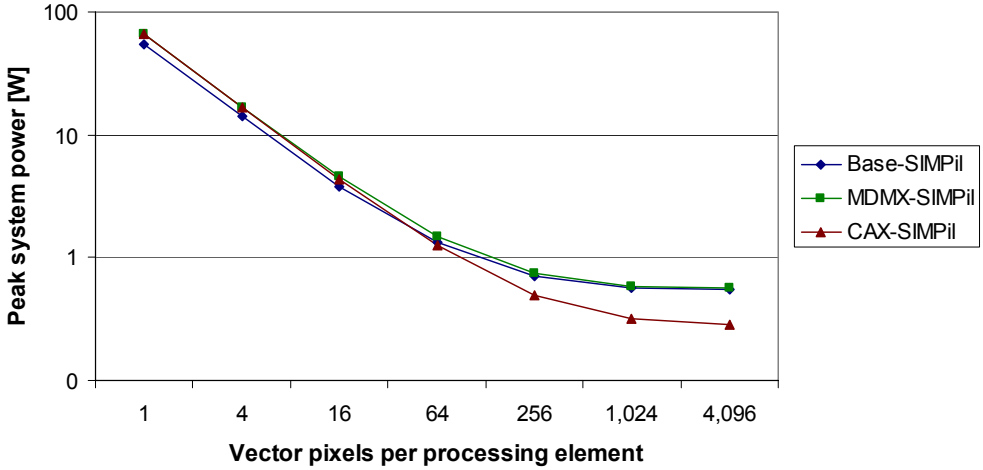


Figure 52. Peak system power versus VPPE.

Figures 53 and 54 present additional data showing the distribution of each functional unit's area and power, respectively, for SIMPil with MDMX and CAX, normalized to the baseline configuration. For VPPEs at or above 64, CAX drastically reduces both system area and power over the baseline configuration because of a large reduction in local memory. Below this point, however, CAX requires higher system area and power than the baseline since the area overhead of the CAX execution unit is more

significant than the benefit of the reduced local memory area. MDMX, however, increases both system area and power for all the configurations. These system areas and powers are combined with application simulations to determine both area and energy efficiency for each case, which is presented next.

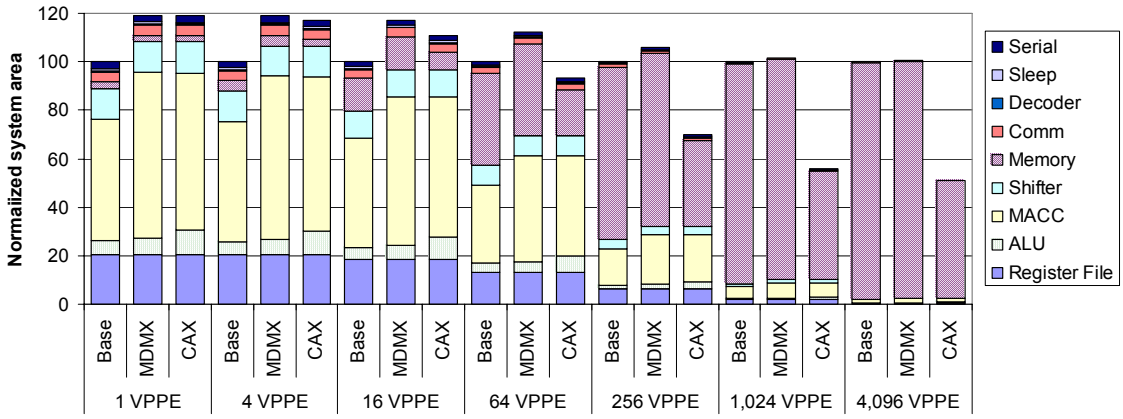


Figure 53. Impact of CAX on system area.

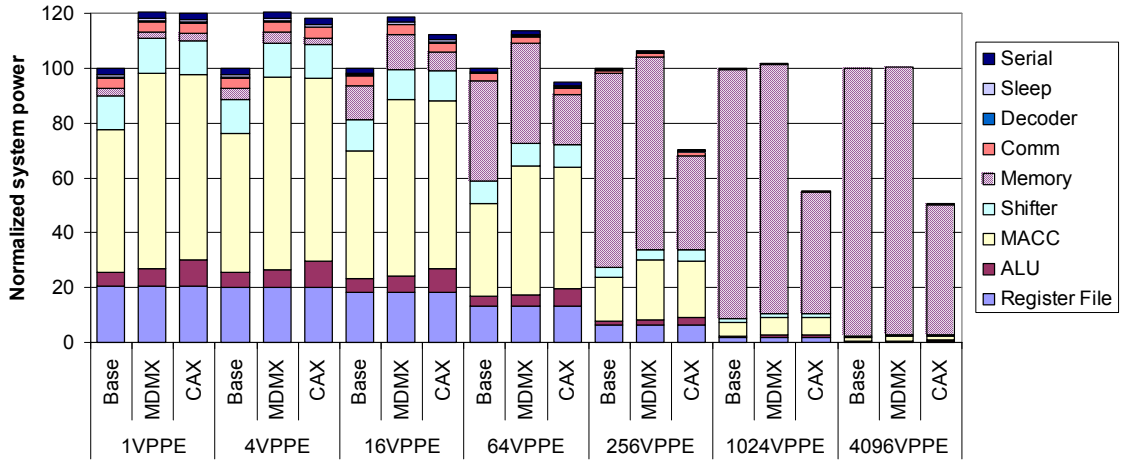


Figure 54. Impact of CAX on system power.

5.6 Experimental Results

Cycle accurate simulation and technology modeling are used to determine performance and efficiency for each architectural configuration for full search vector

quantization (FSVQ). (The parallel FSVQ implementation has been discussed in Section 4.5.1.2.) The execution cycle count, area efficiency, and energy consumption of each case form the basis of the study comparison.

5.6.1 Execution Performance Evaluation Results

This section evaluates the effect of different VPPE ratios on processing performance for each case. The impact of CAX on each VPPE configuration is also presented.

5.6.1.1 Impact of Varying VPPE Ratios on Processing Performance

Figure 55 shows sustained throughputs for different VPPE configurations with and without CAX or MDMX. As expected, the sustained throughput decreases as the VPPE value increases because of less data parallelism (or a decrease in available processing elements).

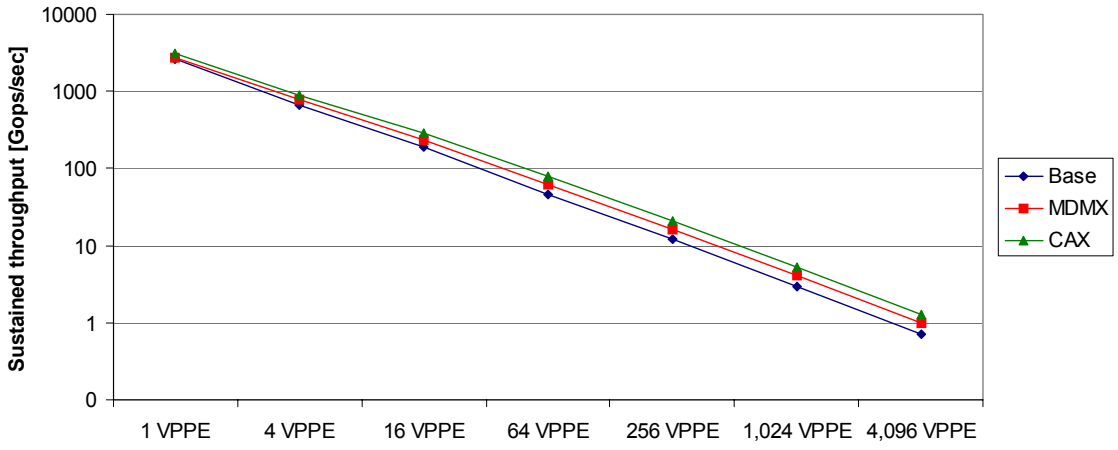


Figure 55. Sustained throughputs for different VPPE configurations with and without CAX or MDMX.

5.6.1.2 Impact of CAX on Different VPPE Configurations

Figure 56 shows the distribution of issued instructions for each VPPE configuration with CAX and MDMX, normalized to the baseline version. Each bar divides the instructions into the arithmetic-logic-unit (ALU), memory (MEM), communication (COMM), PE activity control unit (MASK), image pixel loading (PIXEL), MDMX, and CAX. Results indicate that the instruction count decreases from 29.6% (at VPPE = 1) to 89.4% (at VPPE = 4,096) with CAX, but only 24.8% (at VPPE = 1) to 83.6% (at VPPE = 4,096) with MDMX over the baseline version. As expected, both CAX and MDMX are less effective at reducing vector instructions for VPPEs below 16. This is because high inter-PE communication operations are involved that are not affected by CAX or MDMX.

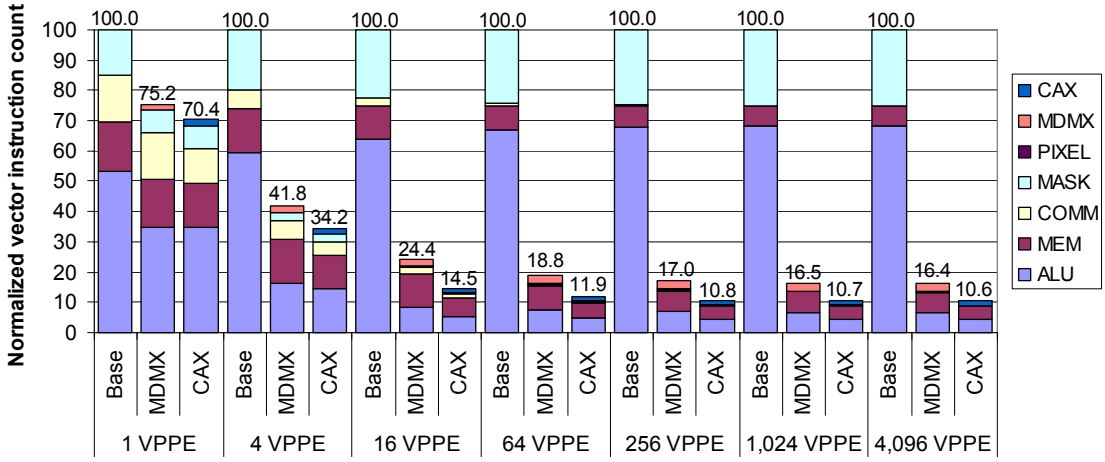


Figure 56. Issued vector instructions for each VPPE configuration with MDMX and CAX, normalized to the baseline version.

Figure 57 presents additional data showing speedups for each VPPE configuration with CAX and MDMX, normalized to the baseline performance. CAX outperforms

MDMX over all VPPE configurations in speedup since CAX consistently reduces more instructions required for the program, indicating 1.4× (at VPPE = 1) to 9.2× (at VPPE = 4,096) with CAX, but only 1.3× (at VPPE = 1) to 6.1× (at VPPE = 4,096) with MDMX over the baseline version.

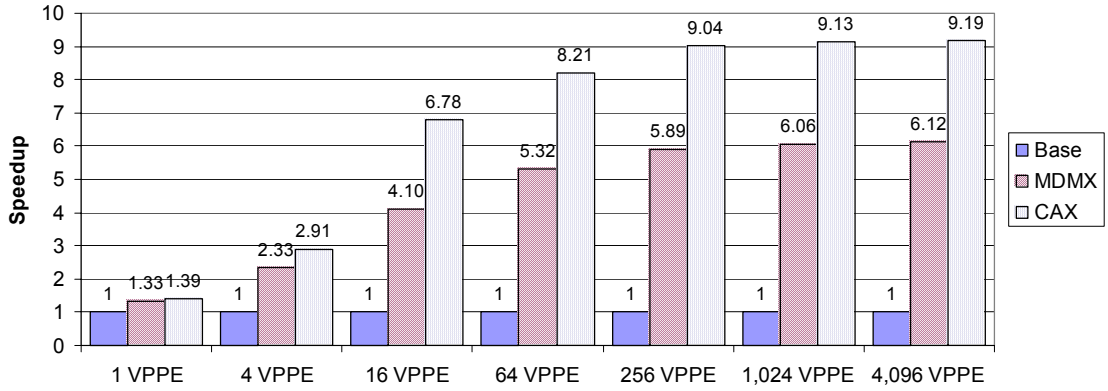


Figure 57. Speedups of each VPPE configuration with CAX and MDMX, normalized to the baseline performance.

CAX also reduces PE idle cycles from 6% (at VPPE = 1) to 26% (at VPPE = 4,096) over the baseline version, shown in Figure 58. This is because CAX compare instructions allow multiple conditional (MASK) instructions with one equivalent CAX instruction, reducing PE idle cycles based on the local information. As with the issued vector instruction count, CAX is less effective at reducing PE idle cycles for VPPEs below 16 because of high inter-PE communication operations that are not affected by CAX. Interestingly, MDMX reduces more PE idle cycles than CAX for all the VPPE configurations. This is because CAX reduces an additional large number of PE active cycles without a proportional decrease in the PE idle cycles. Table 19 summarizes all simulation results. The next two sections evaluate energy- and area-related performance for each case.

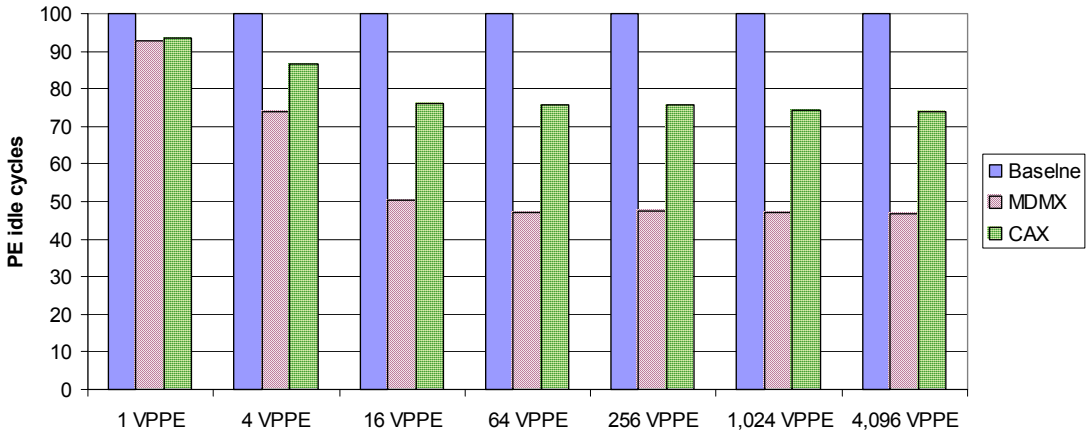


Figure 58. PE idle cycles for each VPPE configuration with CAX and MDMX, normalized to the baseline version.

Table 19. Application performance for each VPPE configuration with and without MDMX or CAX running at 50 MHz.

# of VPPE (# of PE)	ISA	Vector Instruction	Scalar Instruction	System Utilization [%]	Execution Time [msec]	Sustained Throughput [Gops/sec]
1 VPPE (65,536 PEs)	Baseline	34,101	14,873	80.8	0.68	2,646
	MDMX	25,653	14,873	82.1	0.51	2,798
	CAX	24,014	11,609	82.0	0.48	3,116
4 VPPE (16,384 PEs)	Baseline	59,392	18,731	82.8	1.19	678
	MDMX	24,832	18,731	87.3	0.50	789
	CAX	20,302	13,343	85.1	0.41	873
16 VPPE (4,096 PEs)	Baseline	183,860	20,755	92.0	3.68	188
	MDMX	44,850	20,755	96.0	0.90	235
	CAX	26,602	12,259	93.9	0.53	285
64 VPPE (1,024 PEs)	Baseline	684,689	49,707	92.3	13.69	47
	MDMX	128,657	49,707	96.4	2.57	62
	CAX	81,361	28,203	94.2	1.63	79
256 VPPE (256 PEs)	Baseline	2,674,449	164,483	92.0	53.49	12
	MDMX	454,417	164,483	96.2	9.09	16
	CAX	287,761	91,779	94.0	5.76	21
1,024 VPPE (64 PEs)	Baseline	10,634,161	623,469	92.0	212.68	2.9
	MDMX	1,754,033	623,469	96.2	35.08	4.1
	CAX	1,132,513	347,013	94.0	22.65	5.2
4,096 VPPE (16 PEs)	Baseline	42,464,544	2,455,523	91.9	849.29	0.7
	MDMX	6,944,032	2,455,523	96.2	138.88	1.0
	CAX	4,488,368	1,364,907	94.0	89.77	1.3

5.6.2 Area-Related Evaluation Results

Figure 59 presents area efficiency for each case. All three versions achieve their maximum area efficiency at VPPE = 16 due to the inherent definition of the FSVQ program. For VPPEs above 16, the area efficiency decreases almost linearly because the number of operations to perform the task increases more rapidly with VPPE than the area level at which local memory area dominates.

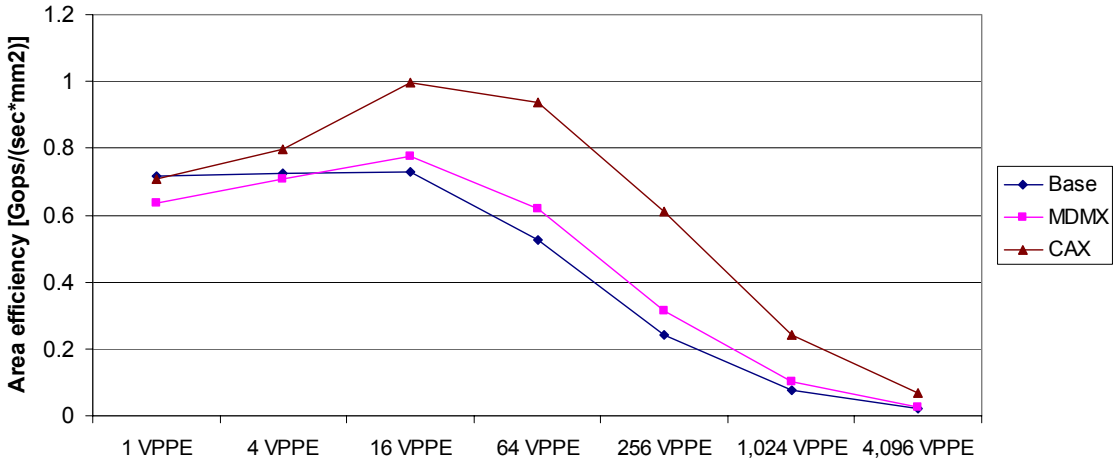


Figure 59. Area efficiency versus VPPE.

5.6.3 Energy-Related Evaluation Results

Figure 60 presents energy consumption for each VPPE configuration with MDMX and CAX, normalized to the baseline version. Each bar divides the energy consumption into the functional unit (FU, combines ALUs, Barrel Shifter, and MACC), storage (combines Register file and Memory), and others (combines Comm., Sleep, Serial, and Decoder) categories. The results indicate that energy consumption for each program is reduced from 26% (at VPPE = 1) to 89% (at VPPE = 4,096) with CAX, but only 24% (at VPPE = 1) to 84% (at VPPE = 4,096) with MDMX over the baseline. For

VPPEs below 16, both MDMX and CAX are less efficient at reducing energy consumption because of the smaller reduction rate in the instruction count.

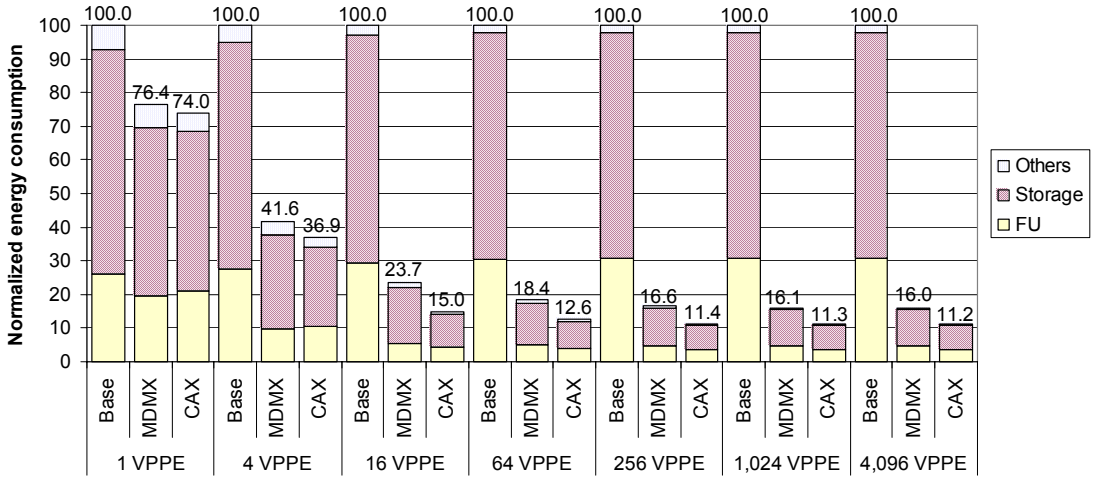


Figure 60. Energy consumption for each VPPE configuration with CAX and MDMX, normalized to the baseline version.

5.7 Conclusion

Reconfigurable silicon area usage within an integrated pixel processing array is a key issue for focal-plane SIMD architectures because of limited chip resources and varying application requirements. In this regard, this chapter has explored the effects of varying the VPPE ratio (number of vector pixels mapped to each processor within a SIMD architecture). Moreover, the impact of CAX on each VPPE configuration has been evaluated to identify the most efficient grain size for a specified SIMD array and implementation technology. Experimental results using architectural and workload simulation indicate that CAX outperforms MDMX for all of the VPPE configurations for full search vector quantization in terms of processing performance, area efficiency, and energy reduction. Results also suggest that high processing performance with the lowest cost is achieved at VPPE = 16 with CAX.

CHAPTER 6

CONCLUSION AND FUTURE WORK

This dissertation has addressed application-, architecture-, and technology-level issues in existing processing systems to provide efficient processing of multimedia in many, or ideally all, of its forms. In particular, this dissertation has explored color imaging for multimedia while focusing on two architectural enhancements for embedded color video and still-image processing: (1) a pixel-truncation technique and (2) a color-aware multimedia instruction set extension (CAX) for embedded multimedia systems. Unlike typical subsampling techniques (e.g., 4:2:2 and 4:2:0) used in image and video compression applications, the pixel-truncation technique reduces information contents in individual pixel word sizes rather than in each dimension while inheriting the chrominance components (Cb and Cr) of the luminance (Y). Thus, this technique significantly reduces the bandwidth and memory required to transport and store color images without a perceivable distortion of color while maintaining the pixel storage format of vector processing in which each pixel computation is simultaneously performed on 3-D color components. Employing the reduced pixel format, CAX supports parallel operations on two-packed, truncated 16-bit YCbCr data in a 32-bit datapath processor, providing greater concurrency and efficiency for processing color image sequences. Thus, CAX, coupled with the pixel-truncation technique, enables higher degrees of parallelism and performance required by emerging imaging applications.

This dissertation has presented the impact of CAX on performance and efficiency with respect to color imaging applications in three major processor architectures: dynamically scheduled (superscalar), statically scheduled (VLIW), and embedded SIMD array processors. Results from the research presented in this dissertation are summarized in the rest of this chapter along with future research directions.

6.1 Summary of Results

6.1.1 Exploring Color Imaging for Multimedia

This research explored color imaging for multimedia to provide new opportunities that define an efficient architecture for embedded multimedia systems. Several color specification models were evaluated to identify the most suitable color space that achieves a natural extension of the imaging operation. In addition, the use of color information in multimedia applications was investigated using a vector approach, improving the accuracy of the process and overall image quality. Furthermore, several color representations with varying pixel word sizes were evaluated to determine the most efficient representation in terms of storage requirements and color accuracy, In particular, a 16-bit (6:5:5) YCbCr representation was examined for reduced-memory, embedded video processing. The 16-bit YCbCr representation reduces the average per pixel word storage requirements by 33% when compared to the baseline 24-bit YCbCr format. Overall video quality remains high, and color imaging applications continue to perform well using the reduced pixel format.

6.1.2 Utilizing Color Subword Parallelism in Superscalar ILP Processors

A new color-aware multimedia extension (CAX) for dynamically scheduled superscalar processors was presented to support color imaging applications. Unlike typical multimedia extensions, CAX obtains substantial performance and code density improvements through direct support of color data processing. Rather than depending solely on generic subword parallelism, CAX supports parallel operations on two-packed, quantized 16-bit YCbCr data in a 32-bit datapath processor, providing greater concurrency and efficiency for processing color image sequences. The key findings follow. CAX achieves a speedup ranging from 3× to 5.8× over the baseline performance on a dynamically scheduled, 4-way issue superscalar processor. This is contrast to MDMX (a representative MIPS multimedia extension), which achieves a speedup of only 1.6× to 3.2× over the baseline. CAX also outperforms MDMX in energy reduction (68% to 83% reduction with CAX, but only 39% to 69% reduction with MDMX over the baseline version). Furthermore, CAX exhibits higher relative performance for low-issue rates. These results demonstrate that CAX is an ideal candidate for embedded multimedia systems in which high issue rates and out-of-order execution are too expensive.

Performance improved by CAX was further enhanced through loop unrolling (LU) that reorganizes and reschedules the loop body. LU provides an additional performance gain of 4%, 19%, and 21% for the baseline, MDMX, and CAX versions, respectively. These results demonstrate that the CAX plus LU technique has the potential to provide the higher degrees of parallelism and performance required by emerging imaging applications.

6.1.3 Implementation and Evaluation of the Color-Aware Instruction Set for Low-Memory, Embedded Video Processing in Data Parallel Architectures

The CAX instruction set was implemented and evaluated for color imaging applications on a representative SIMD array architecture. CAX harnesses parallelism within the human perceptual color space (e.g., YCbCr). In addition, CAX's ability to reduce data format size reduces system cost. The key findings are the following.

- CAX outperforms MDMX across all the selected programs in speedup (5.2× to 8.8× with CAX, but only 3× to 5× with MDMX over the baseline performance) on the same data parallel SIMD execution platform.
- CAX also outperforms MDMX in both area efficiency (a 75% increase versus a 25% increase) and energy efficiency (a 75% increase versus a 24% increase), resulting in better component utilization and sustainable battery life.
- Furthermore, CAX improves the performance and efficiency with a mere 3% increase in the system area and a 5% increase in the system power, while MDMX requires a 14% increase in the system area and a 16% increase in the system power.

6.1.4 Analytically Determining Optimal Grain Sizes in Embedded SIMD Architectures

Reconfigurable silicon area usage within an integrated pixel processing array is a key issue for focal-plane SIMD imaging architectures because of limited chip resources and varying application requirements. The effects of varying the VPPE ratio (number of vector pixels mapped to each processor within a SIMD architecture) on performance and efficiency were evaluated for a specified PE architecture and implementation technology.

Moreover, the impact of CAX on each VPPE configuration was evaluated to identify the most efficient PE granularity that delivers required performance with the lowest cost and the longest battery life. Experimental results for a case study, full search vector quantization, indicate that the VPPE ratio at 16 with CAX provides the most efficient operation for the specified workload.

6.1.5 Static versus Dynamic Scheduling

The performance of static versus dynamic architectures with and without CAX or MDMX was compared to determine whether static or dynamic scheduling is more desirable for color imaging applications. Experimental results through a common simulation framework indicate that the dynamic approach with a four-way issue achieves an average speedup of 2.7× over the static approach with a four-way issue. This is because the static approach is limited by the basic block scheduling algorithm, and the static code schedules are poorly adapted to the run-time conditions of the processor. CAX achieves an additional speedup of 7.6×, while MDMX achieves an additional speedup of only 2.7×.

6.2 Future Research Directions

The research presented in this dissertation is the first to explore and evaluate color imaging for multimedia with novel color-aware multimedia instruction sets in various processor architectures including superscalar, VLIW, and embedded SIMD imaging processors. While a comprehensive evaluation regarding application-, architecture-, and

technology-level issues for supporting color imaging applications has been provided in this dissertation, a number of interesting issues exist for future research.

6.2.1 Color Imaging Metrics and Cost Models

- Evaluate more color space models for identifying the most advantageous color space that achieves the most effective results in color image processing.
- Develop reliable quality metrics for visual performance evaluation because, in many cases, objective image quality measures, such as the mean square error (MSE), the mean absolute error (MAE), and signal-to-noise ratio (SNR), do not provide an accurate or even correct measure of the actual visual quality degradation.
- Develop hardware implementation cost models for several color representations with and without the pixel-truncation technique with respect to the target applications to analyze the implementation efficiency.

6.2.2 An In-depth Analysis of the CAX Instruction Set

- Perform an in-depth analysis of CAX with completed video-processing applications, such as MPEG and H.26L. This will be performed in the context of various processor architectures, ranging from fully custom to fully programmable architectures (e.g., ASICs, superscalar, VLIW, and embedded media processors). This will likely result in adding new instructions (in particular, those performing complex operations) for the completion of the CAX instruction set.

- Compare CAX with a wider ranger of multimedia extensions, industrial as well as those proposed in academic research, while extending the datapath by 64 bits.
- Explore compiler support for extracting color subword parallelism from high level language programs to overcome tedious hand optimization and/or special programming libraries.

6.2.3 Adaptable and Scalable Architectures

- Extend the analysis for variable VPPE mappings to a variety of color imaging applications. This will provide accurate database (e.g., performance, area efficiency, and energy efficiency) for each VPPE configuration.
- Develop heuristic techniques for traversing the design space and extracting both data-level parallelism (DLP) and subword parallelism from a high level language to automatically analyze various workloads. Continued advances in multimedia computing will rely on architecture scalability and adaptability.

APPENDIX A

STATIC VERSUS DYNAMIC SCHEDULING

This appendix compares the performance of static and dynamic architectures with and without CAX or MDMX to determine whether static or dynamic scheduling is more desirable for color imaging applications. All the simulations are conducted through a common simulation framework. Experimental results using the SimpleScalar-based simulator and a retargeting tool indicate that the dynamic approach with a four-way issue achieves an average speedup of 2.7× over the static approach with a four-way issue. CAX achieves an additional speedup of 7.6×, but MDMX achieves an additional speedup of only 2.7×.

Simulation Methodology

Figures 61(a) and (b) present methodology frameworks for dynamically- and statically-scheduled programs, respectively. The Simplescalar-based simulator [2] is used to profile execution statistics for the three different versions (e.g., baseline, MDMX, and CAX) of both static and dynamic programs. For static programs, however, a modified retargeting tool [11] is also used to retarget portable ISA (PISA) assembly code into PISA-derived code amenable for statically-scheduled simulations on the Simplescalar-based simulator with the out-of-order execution capability disabled. Since existing tools, such as a PISA assembler, linker, and binary loader, can be immediately used without modifications, the simulation process is simplified.

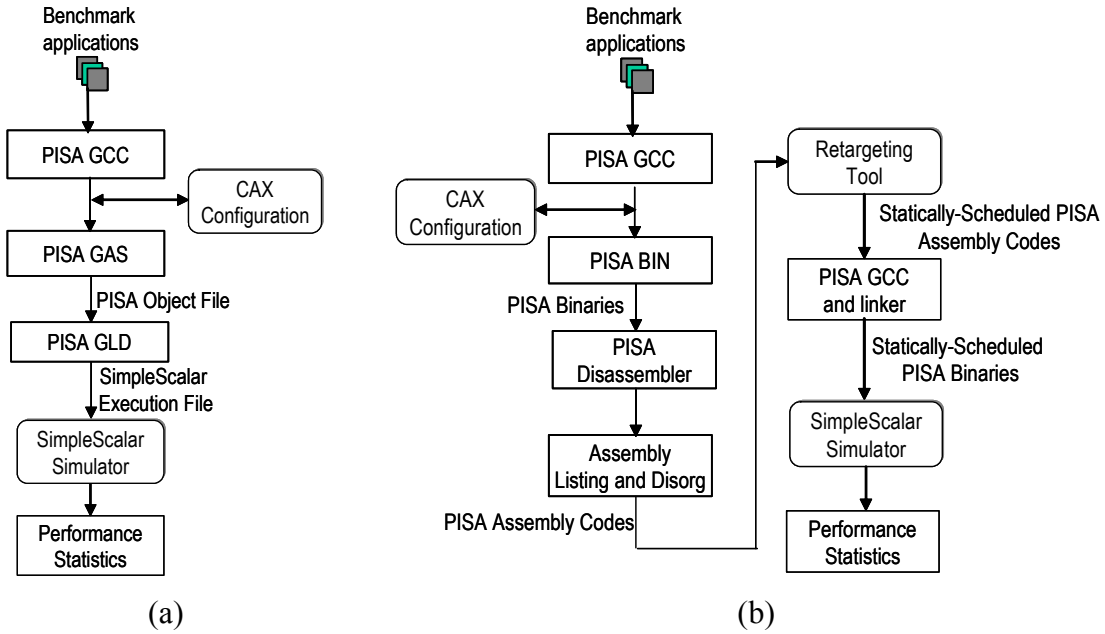


Figure 61. Methodology frameworks: (a) dynamically-scheduled simulations and (b) statically-scheduled simulations.

The MDMX and CAX versions of the programs are created by replacing fragments of the baseline assembly language with ones containing MDMX and CAX instructions. The three different versions of each program have the same parameters, data sets, and calling sequences. Since the target platform is an embedded system, operating system interface code (e.g., file system access) is not included in this study. In the experiment, five color imaging applications (e.g., VSobel, SMF, VMF, VQ, and FSVBMA), summarized in Table 4, are executed on the Simplescalar simulator. Moreover, the same technology and processor configuration, summarized in Table 20, are used.

Table 20. Default processor parameters.

Parameter	Value
Fetch /decode/issue/commit width	4 instructions/cycle
intALU/intMUL/fpALU/fpMUL/Mem	4/2/2/1/4
RUU (window) size	16 instructions
LSQ (Load Store Queue)	8 instructions
Branch Predictor	Combined predictor (1K entries) of bimodal predictor (4K entries) table and 2-level predictor (2-bit counters and 10-bit global history)
L1 D-cache	128-set, 4-way, 32-byte line, LRU, 1-cycle hit, total of 16 KB
L1 I-cache	512-set, direct-mapped 32-byte line, LRU, 1-cycle hit, total of 16 KB
L2 unified cache	1,024-set, 4-way, 64-byte line, LRU, 6-cycle hit, total of 256 KB
Memory latency (memory width)	50 cycles for first chunk, 2 thereafter (64 bits)
Instruction TLB	16-way, 4,096 byte page, 4-way, LRU, 30 cycle miss penalty
Data TLB	32-way, 4,096 byte page, 4-way, LRU, 30 cycle miss penalty

Experimental Results

Figure 62 shows execution performance (speedup in executed cycles) for two variations of the baseline architecture, each without subword parallelism, with MDMX, and with CAX. The two architecture variations are (1) static and four-way issue and (2) dynamic and four-way issue. All the execution performance is normalized to the baseline static performance without subword parallelism. The dynamic approach without subword parallelism achieves a speedup ranging from 2.6× to 3× (an average speedup of 2.7×) over the baseline static performance. This is because the static approach is limited by the basic block scheduling algorithm, and the static code schedules are poorly adapted to the run-time conditions of the processor. CAX achieves an additional speedup of 7.6×, but MDMX achieves an additional speedup of only 2.7×. This is because CAX supports more color data elements in a register while processing these separate color elements in parallel.

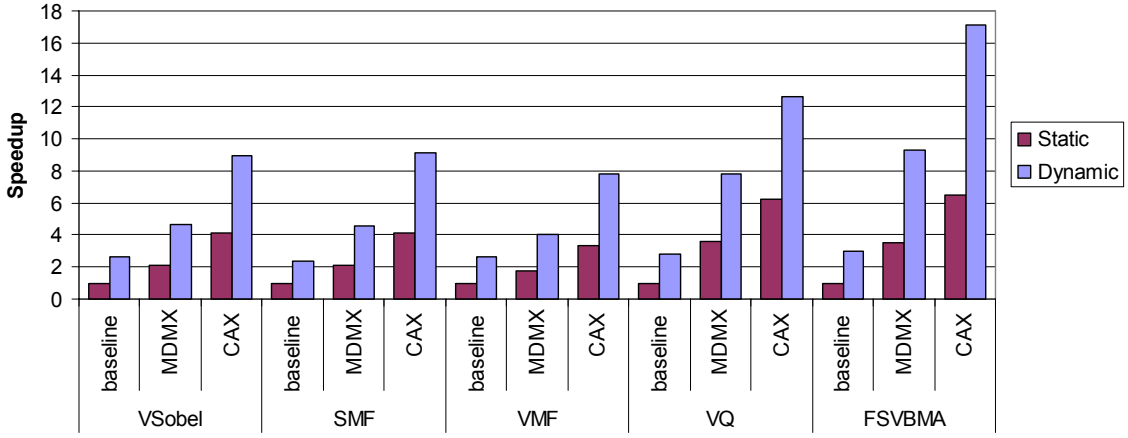


Figure 62. Speedups for the dynamically scheduled superscalar processor with and without MDMX or CAX over the baseline static performance without subword parallelism.

Conclusion

Although static architectures (e.g., VLIW and DSP) have been exclusively used in existing media processors because of low cost and power, they will not meet the higher demands for performance required by emerging multimedia applications. Thus, the dynamic aspects of processing become more pronounced. This appendix has compared the performance of dynamic versus static approaches with and without subword parallelism to determine which approach is more desirable for color imaging applications. Experimental results using a common simulation framework indicate that the dynamic approach with a four-way issue achieves an average speedup of 2.7× over the static performance with a four-way issue. CAX achieves an additional speedup of 7.6×.

APPENDIX B

CAX: A COLOR-AWARE INSTRUCTION SET

CAX applied to current microprocessor ISAs is targeted to accelerating color image- and video-processing applications. Combined with a 32-bit datapath processor, CAX supports parallel operations on two-packed, quantized 16-bit (6:5:5) YCbCr data, providing greater concurrency and efficiency for processing color image sequences. Moreover, CAX employs 128-bit color-packed accumulators that provide solutions to overflow and other issues caused by packing data as tightly as possible by implicit width promotion and adequate space.

CAX Instructions (grouped by functionality)

Table 21 lists all the CAX instructions available. These CAX instructions exploit color subword parallelism within the context of three major processor architectures: dynamically scheduled (superscalar), statically scheduled (VLIW), and embedded SIMD array processors.

Table 21: CAX instruction descriptions.

Instructions	Description
Parallel ALU Instructions	
C_PADD_SW	Parallel Addition – Signed Wrap Around
C_PADD_UW	Parallel Addition – Unsigned Wrap Around
C_PADD_SS	Parallel Addition – Signed Saturation
C_PADD_US	Parallel Addition – Unsigned Saturation
C_PSUB_SW	Parallel Subtraction – Signed Wrap Around
C_PSUB_UW	Parallel Subtraction – Unsigned Wrap Around
C_PSUB_SS	Parallel Subtraction – Signed Saturation
C_PSUB_US	Parallel Subtraction – Unsigned Saturation
C_PAVG_U	Parallel Average – Unsigned

Table 21: (Continued)

Parallel Compare Instructions	
C_PCMP_EQ	Parallel Compare Equal
C_PCMP_NE	Parallel Compare Not Equal
C_PCMP_LT	Parallel Compare Less Than – Signed
C_PCMP_LE	Parallel Compare Less Equal – Signed
C_PCMP_GT	Parallel Compare Greater Than – Signed
C_PCMP_GE	Parallel Compare Greater Equal – Signed
C_PCMP_LT_U	Parallel Compare Less Than – Unsigned
C_PCMP_LE_U	Parallel Compare Less Equal – Unsigned
C_PCMP_GT_U	Parallel Compare Greater Than – Unsigned
C_PCMP_GE_U	Parallel Compare Greater Equal – Unsigned
C_PMAX_U	Parallel Maximum – Unsigned
C_PMIN_U	Parallel Minimum – Unsigned
C_PCMOV	Parallel Conditional Move
Permute Instructions	
C_MIX_L	Mix Left
C_MIX_R	Mix Right
C_ROTATE_R	Rotate Right
C_BCAST_SS	Broadcast – Signed Saturation
Special-Purpose Instructions	
C_PADACC_U_S	Parallel Absolute Differences Accumulation with Unsigned Values – Signed
C_PMACC_U_S	Parallel Multiply and Accumulation with Unsigned Value – Signed
C_PMACC_U_S_S	Parallel Multiply and Accumulation with U/S Values – Signed
C_ZACC	Zero Accumulator
C_RACL	Read the Least Significant 32 bits of an Accumulator
C_RACS	Read the Second Significant 32 bits of an Accumulator
C_RACT	Read the Third Significant 32 bits of an Accumulator
C_RACH	Read the Most Significant 32 bits of an Accumulator

Parallel Add Instructions

C_PADD_SW	Parallel Addition – Signed Wrap Around
C_PADD_UW	Parallel Addition – Unsigned Wrap Around
C_PADD_SS	Parallel Addition – Signed Saturation
C_PADD_US	Parallel Addition – Unsigned Saturation

Format:
```
c_padd_sw   Rd,Rs1,Rs2
c_padd_uw   Rd,Rs1,Rs2
c_padd_ss   Rd,Rs1,Rs2
c_padd_us   Rd,Rs1,Rs2
```

Description: Rd[i]← Rs1[i] + Rs2[i]

The parallel add instructions add the sub-elements of Rs1 from the corresponding sub-elements of Rs2. The results are then written to Rd.

The c_padd_sw instruction uses signed wrap around; the c_padd_uw instruction uses unsigned wrap around; the c_padd_ss instruction uses signed saturation; and the c_padd_us instruction uses unsigned saturation. For saturated arithmetic operations, overflows and underflows clamp to the largest or smallest value before writing to the destination register.

Parallel Subtract Instructions

C_PSUB_SW	Parallel Subtraction – Signed Wrap Around
C_PSUB_UW	Parallel Subtraction – Unsigned Wrap Around
C_PSUB_SS	Parallel Subtraction – Signed Saturation
C_PSUB_US	Parallel Subtraction – Unsigned Saturation

Format:
```
c_psub_sw   Rd,Rs1,Rs2
c_psub_uw   Rd,Rs1,Rs2
c_psub_ss   Rd,Rs1,Rs2
c_psub_us   Rd,Rs1,Rs2
```

Description: Rd[i]← Rs1[i] - Rs2[i]

The parallel subtract instructions subtract the sub-elements of Rs2 from the corresponding sub-elements of Rs1. The results are then written to Rd.

The c_psub_sw instruction uses signed wrap around; the c_psub_uw instruction uses unsigned wrap around; the c_psub_ss instruction uses signed saturation; and the c_psub_us instruction uses unsigned saturation. For saturated arithmetic operations, overflows and underflows clamp to the largest or smallest value before writing to the destination register.

Parallel Average Instructions

C_PAVG_U	Parallel Average – Unsigned

Format: c_pavg_u Rd,Rs1,Rs2

Description: Rd[i]← round(avg(Rs1[i], Rs2[i]))

The parallel average instruction adds the sub-elements of Rs1 to the corresponding sub-elements of Rs2. The sums are then shifted right by one bit. (If each sum has a positive value, the most significant bit becomes 0 during shifting to the right. Otherwise, the most significant bit becomes 1.) The shifted results are then written to Rd, in which the least significant bit of each resulting subword is obtained by a logical *or* operator of the two least significant bits of the shifted sums. These instructions are useful for blending algorithms.

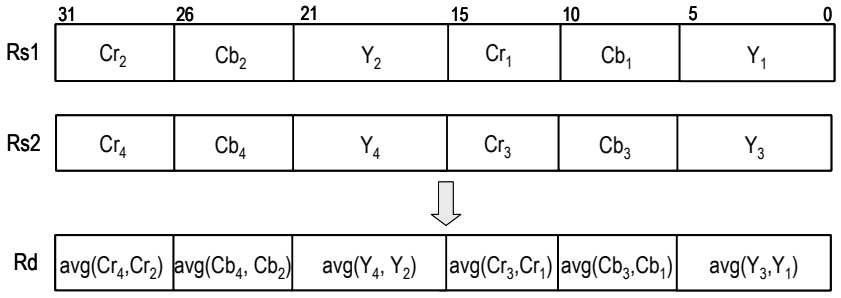


Figure 63. An example of a parallel average instruction.

Parallel Compare Instructions

C_PCMP_EQ	Parallel Compare Equal
C_PCMP_NE	Parallel Compare Not Equal
C_PCMP_LT	Parallel Compare Less Than – Signed
C_PCMP_LE	Parallel Compare Less Equal – Signed
C_PCMP_GT	Parallel Compare Greater Than – Signed
C_PCMP_GE	Parallel Compare Greater Equal – Signed
C_PCMP_LT_U	Parallel Compare Less Than – Unsigned
C_PCMP_LE_U	Parallel Compare Less Equal – Unsigned
C_PCMP_GT_U	Parallel Compare Greater Than – Unsigned
C_PCMP_GE_U	Parallel Compare Greater Equal – Unsigned
C_PCMOV	Parallel Conditional Move

Format: c_pcmp_fuc Rd,Rs1,Rs2 (fuc : EQ, NE, LT, LE, GT, GE)
c_pcmov Rd,Rs1,Rs2

Description: Rd[i]← (Rs1[i] cond Rs2[i])

The c_pcmp_fuc instructions compare pairs of the sub-elements in Rs1 and Rs2 and write the results to Rd. Depending on the instructions, the results are varied for each sub-element comparison. The c_pcmp_eq instruction, for example, compares pairs of the sub-elements in Rs1 and Rs2 and writes a bit string of 1s for true comparison results and 0s for false comparison results to Rd.

In the c_pcmov instruction, the packed operands of Rd are

1) the sub-elements of Rs1 if each sub-element of Rs2 is equal to all 1s,
2) Rd otherwise.

Parallel Max/Min Instructions

C_PMAX_U	Parallel Maximum – Unsigned
C_PMIN_U	Parallel Minimum – Unsigned

Format: c_pmax_u Rd,Rs1,Rs2
c_pmin_u Rd,Rs1,Rs2

Description: Rd[i]← max(Rs1[i], Rs2[i]) or Rd[i]← min(Rs1[i], Rs2[i])

The c_pmax_u instruction compares pairs of the unsigned sub-elements in the two source registers and outputs the maximum values to the destination register.

The c_pmin_u instruction compares pairs of the unsigned sub-elements in the two source registers and outputs the minimum values to the destination register.

Permute Instructions

C_MIX_L	Mix Left
C_MIX_R	Mix Right
C_ROTATE_R	Rotate Right
C_BCAST_SS	Broadcast – Signed Saturation

Format:
```
c_mix_l     Rd,Rs1,Rs2
c_mix_r     Rd,Rs1,Rs2
c_rotate_r  Rd,Rs1,Imm
c_bcast_ss  Rd,Rs1,Imm
```

Description: Rd[i]← select(i,Rs1[i], Rs2[i])

The mix instructions mix the sub-elements of Rs1 and Rs2 into the operands of Rd.

The rotate instruction rotates the sub-elements to the right by an immediate value.

The broadcast instruction writes the selected sub-elements of Rs1 by an immediate indicator to all the sub-elements of Rd.

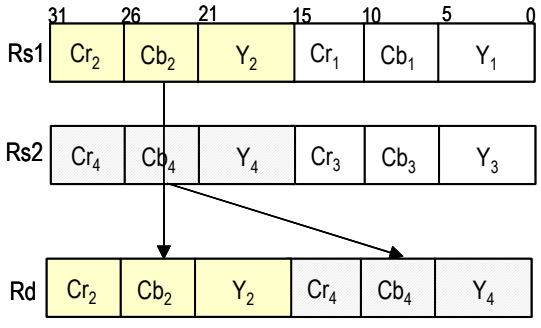


Figure 64. An example of a mix left instruction.

Parallel Absolute Differences Accumulation Instructions

C_PADACC_U_S	Parallel Absolute-Differences-Accumulate with Unsigned Values – Signed

Format: c_padacc_u_s Rd,Rs1,Rs2

Description: acc[i]← acc[i] + abs(Rs1[i] – Rs2[i]), Rd[i]← abs(Rs1[i] – Rs2[i])

The sum of absolute-distance-accumulate instruction calculates the absolute differences of pairs of the sub-elements in Rs1 and Rs2 while accumulating each result in the accumulator. In the mean time, each absolute result is stored to Rd. This instruction is frequently used by a number of algorithms for motion estimation.

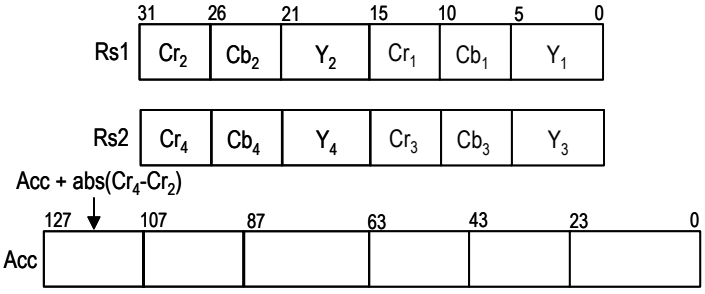


Figure 65. An example of a PADACC instruction.

Parallel Multiply-Accumulate Instructions

C_PMACC_U_S	Parallel Multiply and Accumulation with Unsigned Values – Signed
C_PMACC_U_S_S	Parallel Multiply and Accumulation with U/S Values – Signed

Format: c_pmacc_u_s Acc,Rs1,Rs2
c_pmacc_u_s_s Acc,Rs1,Rs2

Description: acc[i]← acc[i] + abs(Rs1[i] * Rs2[i])

The c_pmacc_u_s instruction multiplies the unsigned sub-elements of Rs1 with the corresponding unsigned sub-elements of Rs2 while accumulating each result in the packed signed operands of the accumulator.

The c_pmacc_u_s_s instruction multiplies the unsigned sub-elements of Rs1 with the corresponding signed sub-elements of Rs2 while accumulating each result in the packed signed operands of the accumulator. These instructions are useful in DSP algorithms that involve computing a vector dot-product, such as digital filtering and convolutions.

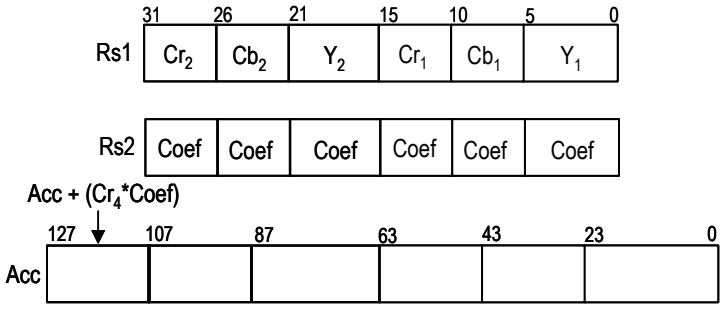


Figure 66. An example of a multiply-accumulate instruction.

ZACC Instructions

C_ZACC	Zero Accumulator

Format: c_zacc Acc(i)

Description: $Acc(i) \leftarrow 0$

The zero accumulator instruction initializes the value of the accumulator to zero.

Examples:

```
c_zacc          acc1          ;; acc1 ← 0
c_pmacc_u_s_s   acc1,r5,r7    ;; acc1_new ← r5 * r7 + 0
c_pmacc_u_s_s   acc1,r5,r7    ;; acc1_new ← r5 * r7 + acc1_old
c_zacc          acc1          ;; acc1 ← 0
```

Read Accumulator Instructions

C_RACL	Read the Least Significant 32 bits of an Accumulator
C_RACS	Read the Second Significant 32 bits of an Accumulator
C_RACT	Read the Third Significant 32 bits of an Accumulator
C_RACH	Read the Most Significant 32 bits of an Accumulator

Format:
```
c_racl   Rd,Acc
c_racs   Rd,Acc
c_ract   Rd,Acc
c_rach   Rd,Acc
```

Description: Rd ← acc{low, mid_left, mid_right, high}

Read either the least significant, second most significant, third most significant, or most significant fourth of bits of the accumulator.

REFERENCES

[1] J. Astola, P. Haavisto, and Y. Neuvo, "Vector median filters," *Proc. of the IEEE*, vol. 78, no. 4, pp. 678-689, April 1990.

[2] T. Austin and D. Burger, "The SimpleScalar Tool Set, Version 2.0," TR-1342, Computer Sciences department, University of Wisconsin, Madison.

[3] B. Barnett, *Handbook of Image Processing*, A. Bovik, ed., Academic Press, 2000.

[4] K. E. Batcher, "Design of a massively parallel processor," *IEEE Trans. Computers C-29*, pp. 836-840, 1980.

[5] A. Bellaouar and M. I. Elmasry, *Low-Power Digital VLSI Design: Circuits and Systems.* Boston: Kluwer Academic, 1995.

[6] R. Bhargava, L. John, B. Evans, and R. Radhakrishnan, "Evaluating MMX technology using DSP and multimedia applications," *Proc. of IEEE/ACM Sym. on Microarchitecture*, pp. 37-46, 1998.

[7] M. Bolotski, R. Armithrajah, W. Chen, "ABACUS: A High Performance Architecture for Vision," in *Proceedings of the International Conference on Pattern Recognition*, 1994.

[8] S. Bond, S. Jung, O. Vendier, M. Brooke, N. M. Jokerst, S. Chai, A. Lopez-Lagunas, and D. S. Wills, "3D stacked Si CMOS VLSI smart pixels using through-Si optoelectronic interconnections," in *Proc. IEEE Lasers and Electro-Optics Soc. Summer Topical Meeting on Smart Pixels*, pp. 27-28, July 1998.

[9] W. J. Bouknight, S. A. Denenberg, D. E. McIntre, J. M. Randall, A. H. Sameh, and D. L. Slotnick, "The Illiac IV system," *Proc. IEEE,* vol. 60, no. 4, pp. 369-388, 1972.

[10] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *Proc. Intl. Symposium on Computer Architecture*, pp. 83-94, June 2000.

[11] S. Bunchua, "Fully distributed register files for heterogeneous clustered microarchitectures," PhD dissertation, Georgia Inst. of Technology, July 2004.

[12] K. Castille, "TMS320C6000 power consumption summary," *Application Report SPRA486B*, Texas Instruments, C6000 Application team, Nov. 1999.

[13] H. H. Cat, A. Gentile, J. C. Eble, M. Lee, O. Verdier, Y. J. Joo, D. S. Wills, M. Brooke, N. M. Jokerst, A. S. Brown, and R. Leavitt, "SIMPil: An OE integrated SIMD architecture for focal plane processing applications," in *Proc. Massively Parallel Processing Using Optical Interconnection (MPPOI-96)*, pp. 44-52, Oct. 1996.

[14] S. M. Chai, T. M. Taha, D. S. Wills, and J. D. Meindl, "Heterogeneous architecture models for interconnect-motivated system design," *IEEE Trans. VLSI Systems*, special issue on system level interconnect prediction, vol. 8, no. 6, pp. 660-670, Dec. 2000.

[15] S. M. Chai, "Real time image processing on parallel arrays for gigascale integration," PhD dissertation, Georgia Inst. of Technology, Nov. 1999.

[16] A. P. Chandrakasan, *Low-power Digital CMOS Design*, Kluwer Academic Publishers, 1995.

[17] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE J. Solid-State Circuits*, vol. 5, no. 2, pp. 140-149, April 1992.

[18] J. Cobal, M. Valero, and R. Espasa, "Exploiting a new level of DLP in multimedia applications," in *Proc. IEEE Intl. Symp. on Microarchitecture (MICRO-32)*, pp. 72-79, Nov. 1999.

[19] *Coding of Moving Pictures and Audio*, ISO/IEC JTC1/SC29/WG11 N3312, 2000.

[20] L. Codrescu, S. P. Nugent, J. D. Meindl, and D. S. Wills, "Modeling technology impact on cluster microprocessor performance," *IEEE Trans. VLSI Systems*, vol. 11, no. 5, pp. 909-920, Oct. 2003.

[21] A color version of this dissertation along with all color images used: http://www.ece.gatech.edu/research/pica/grads/jmkim/thesis/

[22] T. M. Conte, P. K. Dubey, M. D. Jennings, R. B. Lee, A. Peleg, M. Schlansker, P. Song, and A. Wolfe, "Challenges to combining general-purpose and multimedia processors," *IEEE Computer*, vol. 30, no. 12, pp. 33-37, Dec. 1997.

[23] A. Cuhadar, D. Sampson, and A. Downton, "A scalable parallel approach to vector quantization," *Real-Time Imaging*, vol. 2, no. 4, pp. 241-247, Oct. 1996.

[24] K. Diefendorff and R. Dubey, “How multimedia workloads will change processor design,” *IEEE Computer,* vol. 30, no. 9, pp. 43-45, Sept. 1997.

[25] K. Dezhgosha, M. M. Jamali, and S. C. Kwatra, “A VLSI architecture for real-time image coding using a vector quantization based algorithm,” *IEEE Trans. Image Processing*, vol. 40, no. 1, pp. 181-189, 1992.

[26] J. J. Dongarra and A. R. Hinds, “Unrolling loops in Fortran,” *Software-Practice and Experience*, vol. 9, no. 3, pp. 219-226, 1979.

[27] J. C. Eble, “A generic system simulator with novel on-chip cache and throughput models for gigascale integration,” PhD dissertation, Georgia Inst. of Technology, 1998.

[28] J. C. Eble, V. K. De, D. S. Wills, and J. D. Meindl, “A generic system simulator (GENESYS) for ASIC technology and architecture beyond 2001,” in *Proc. of the Ninth Ann. IEEE Intl. ASIC Conf.*, pp. 193-196, Sept. 1996.

[29] E. R. Fossum, “Digital camera system on a chip,” *IEEE Micro,* vol.18, no. 3, pp. 8-15, 1998.

[30] E. R. Fossum, "Architectures for Focal Plane Image Processing," *Optical Engineering*, vol. 28, no. 8, pp. 865-871, 1989.

[31] J. Fridman and Z. Greenfield, “The TigerSHARC DSP architecture,” in *Proc. IEEE/ACM Intl. Sym. on Computer Architecture*, pp. 124-135, May 1999.

[32] J. Fritts, “Architecture and compiler design issues in programmable media processors,” Ph.D. Thesis, Dept. of Electrical Engineering, Princeton University, 2000.

[33] A. Gentile, S. Sander, L. M. Wills, and D. S. Wills, “Impact of pixel to processing ratio in embedded SIMD image processing architectures,” in *Journal of Parallel and Distributed Computing*, vol. 64, no. 11, pp. 1318-1332, Nov. 2004.

[34] A. Gentile and D. S. Wills, “Portable Video Supercomputing,” *IEEE Trans. on Computers*, vol. 53, no. 8, pp. 960-973, Aug. 2004.

[35] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Press, 1992.

[36] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 2nd Ed., Prentice Hall, 2002.

[37] H.-M. Hang and B. G. Haskell, “Interpolative vector quantization of color images,” *IEEE Transactions on Communications*, vol. 36, no. 4, pp. 465-470, April 1988.

[38] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 2003.

[39] M. C. Herbordt, A. Anand, O. Kidwai, R. Sam, and C. C. Weems, “Processor/memory/array size tradeoffs in the design of SIMD arrays for a spatially mapped workload,” in *Proc. IEEE Intl. Workshop on Computer Architecture for Machine Perception*, pp. 12-21, Oct. 1997.

[40] M. J. Irwin, R. M. Owens, "A Two-Dimensional, Distributed Logic Processor," *IEEE Transactions on Computers*, vol. 40, no. 10, pp. 1094-1101, 1991.

[41] M. D. Jennings and T. M. Conte, “Subword extensions for video processing on mobile systems,” *IEEE Concurrency*, vol. 6, no. 3, pp. 13-16, July-Sept. 1998.

[42] B. Juurlink, D. Tcheressiz, S. Vassiliadis, and H. A. G. Wijshoff, “Implementation and evaluation of the complex streamed instruction set,” in *Proc. IEEE Intl. Conf. on Parallel Architectures and Compilation Techniques*, pp. 73-82, Sept. 2001.

[43] J. Kim, S. Ryu, A. Gentile, L. M. Wills, and D. S. Wills, “Impulse noise removal on an embedded, low memory SIMD processor,” in *Proc. of the IEEE Intl. Conf. on Digital Signal Processing*, pp. 1257-1260, July 2002.

[44] J. Kim, L. M. Wills, and D. S. Wills, “Effective detection and elimination of impulse noise for reliable 4:2:0 YCbCr signals prior to compression encoding,” to appear in *Proc. of the IEEE Conf. on Acoustics, Speech, and Signal Processing (ICASSP 05),* March 2005.

[45] J. Kim and D. S. Wills, “Evaluating a 16-bit YCbCr (6:5:5) color representation for low memory, embedded video processing,” in *Proc. of the IEEE Conf. on Consumer Electronics (ICCE’05)*, pp. 181-182, Jan. 2005.

[46] J. Kim and D. S. Wills, “Efficient processing of color image sequences using a color-aware instruction set on mobile systems,” in *Proc. of the IEEE Conf. on Application-Specific Systems, Architectures, and Processors*, pp. 137-149, Sept. 2004.

[47] A. Koschan, “A comparative study on color edge detection,” in *Proc. of the Second Asian Conference on Computer Vision,* vol. III, pp. 574-578, Dec. 1995.

[48] A. Krikelis, I. P. Jalowiecki, D. Bean, R. Bishop, M. Facey, D. Boughton, S. Murphy, and M. Whitaker, "A programmable processor with 4096 processing units for media applications," in *Proc. of the IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, vol. 2, pp. 937-940, May 2001.

[49] T. Kuroda, "Low power CMOS digital design for multimedia processors," in *Proc. of Intl. Conf. on VLSI and CAD*, pp. 359-367, Oct. 1999.

[50] S. C. Kwatra, C. M. Lin, and W. A. Whyte, "An adaptive algorithm for motion compensated color image coding," *IEEE Trams. Commun*., vol. COM-35, pp. 747-754, July 1987.

[51] V. Lappalainen, "Performance of an advanced video codec on a general-purpose processor with media ISA extensions," *IEEE Transactions on Consumer Electronics,* vol. 46, no. 3, pp. 706-716, 2000.

[52] R. B. Lee, "Multimedia extensions for general-purpose processors," *Proc. IEEE Workshop on Signal Processing Systems,* pp. 9-23, 1997.

[53] R. B. Lee, "Subword parallelism with MAX-2," *IEEE Micro,* vol. 16, no. 4, pp. 51-59, Aug. 1996.

[54] R. B. Lee and M. D. Smith, "Media processing: A new design target," *IEEE Micro*, vol. 16, no. 4, pp. 6-9, August 1996.

[55] Y. Y. Lee and J. W. Woods, "Motion vector quantization for video coding," *IEEE Trans. Image Processing*, vol. 4, no. 3, pp. 378-382, 1995.

[56] M. Manohar and J. C. Tilton, "Progressive vector quantization on a massively parallel SIMD machine with application to multispectral image data," *IEEE Trans. on Image Processing*, vol. 5, no. 1, pp. 142-147, Jan. 1996.

[57] MasPar (MP-2) System Data Sheet, MasPar Corp., 1993.

[58] MATLAB Homepage: http://www.mathworks.com/

[59] J. D. Meindl, "Low power microelectronics: Retrospect and prospect," in *Proc. IEEE*, vol. 83, no. 4, pp. 619-635, April 1995.

[60] *MIPS extension for digital media with 3D*, Technical Report http://www.mips.com, MIPS technologies, Inc., 1997.

[61] M. J. Nadenau and J. Reichel, "Opponent color, human vision and wavelets for image compression," in *IS&T/SID Seventh Color Imaging Conference*, pp. 237-242, Nov. 1999.

[62] A. Nakada, T. Shibata, M. Konda, T. Morimoto, and T. Ohmi, "A fully parallel vector-quantization processor for real-time motion-picture compression," *IEEE J. Solid-State Circuits*, vol. 34, no. 6, pp. 822-830, 1999.

[63] H. Nguyen and L. John, "Exploiting SIMD parallelism in DSP and multimedia algorithms using the AltiVec technology," in *Proc. Intl. Supercomputer Conference*, pp. 11-20, June 1999.

[64] T. Nozawa et al., "A parallel vector-quantization processor eliminating redundant calculations for real-time motion picture compression," *IEEE J. Solid-State Circuits*, vol. 35, no. 11, pp. 1744-1751, 2000.

[65] S. Nugent, D. S. Wills, and J. D. Meindl, "A hierarchical block-based modeling methodology for SoC in GENESYS," in *Proc. of the 15th Ann. IEEE Intl. ASIC/SOC Conf.*, pp. 239-243, Sept. 2002.

[66] S. Obereman, G. Favor, and F. Weber, "AMD 3DNow! technology: architecture and implementations," *IEEE Micro,* vol. 19, no. 2, pp. 37-48, 1999.

[67] A. Peleg and U. Weiser, "MMX technology extension to the Intel architecture," *IEEE Micro,* vol.16, no. 4, pp. 42-50, Aug. 1996.

[68] A. Peleg, S. Wilkie, and U. Weiser, "Intel MMX for multimedia PCs," in *Proc. Communications of the ACM*, vol. 40, no. 1, pp. 25-38, Jan. 1997.

[69] K. N. Plataniotis and A. N. Venetsanopoulos, *Color Image Processing and Applications*, Springer Verlag, 2000.

[70] S. K. Raman, V. Pentkovski, and J. Keshava, "Implementing streaming SIMD extensions on the Pentium III processor," *IEEE Micro,* vol. 20, no. 4, pp.28-39, Aug. 2000.

[71] R. Ranganathan, S. Ave, and N. Jouppi, "Performance of image and video processing with general-purpose processors and media ISA extensions," *Proc. of IEEE/ACM Sym. on Computer Architecture*, pp. 124-135, 1999.

[72] A. L. Rosenberg, "Three-dimensional integrated circuits," *VLSI systems and computations*, H. T. Kung, R. F. Sproull and G. L. Steele (eds.), Computer Science Press, Rockville, MD, pp. 69-80, 1981.

[73] J. Scharcanski and A. N. Venetsanopoulous, "Edge detection of color images suing directional operators," *IEEE Trans. on Circuit and Systems for Video Technology*, vol. 7, no. 2, pp. 397-401, April 1997.

[74] Semiconductor Industry Assoc., The International Technology Roadmap for Semiconductors, 2003, Available at http://public.itrs.net.

[75] R. Sites, Ed., *Alpha Reference Manual*, Burlington, MA: Digital, 1992.

[76] N. T. Slingerland and A. J. Smith, "Measuring instruction sets for general purpose microprocessors: A survey," University of California at Berkeley Technical Report CSD-00-1122, 2000.

[77] N. T. Slingerland and A. J. Smith, "Measuring the performance of multimedia instruction sets," *IEEE Transactions on Computers*, vol. 51, no. 11, pp. 1317-1332, 2002.

[78] J. Suh and V. K. Prasanna, "An efficient algorithm for out-of-core matrix transposition," *IEEE Trans. on Computers*, vol. 51, no. 4, pp. 420-438, April 2002.

[79] V. Tiwari, S. Malik, and A. Wolfe, "Compilation techniques for low energy: An overview," in *Proc. Symp. Low Power Electron.*, pp. 38-39, Oct. 1994.

[80] M. Tremblay, J. M. O'Connor, V. Narayanan, and L. He, "VIS speeds new media processing," *IEEE Micro*, vol. 16, no. 4, pp. 10-20, Aug. 1996.

[81] "Connection machine model CM-2 technical summary," Thinking Machines Corp., version 51, May 1989.

[82] TMS320C64x DSP Technical Brief.
Available: http://www.ti.com/sc/docs/products/dsp/c6000/c64xmptb.pdf

[83] L. W. Tucker and G. G. Robertson, "Architecture and applications of the connection machine," *IEEE Computer*, vol. 21, no. 8, pp. 26-38, 1988.

[84] M. J. Vrhel, "Color imaging: current trends and beyond," in *Proceedings of the IEEE International Conference on Image Processing*, vol. 1, pp. 513-516, Sept. 2000.

[85] Y. Wang, J. Ostermann, and Y-Q. Zhang, *Video Processing and Communications*, Prentice Hall, 2002.

[86] S. Weiss and J. E. Smith, "A study of scalar compilation techniques for pipelined supercomputers," in *Proc. Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 105-109, 1987.

[87] C. C. Yang, "Effects of coordinate systems on color image processing," MS Thesis, University of Arizona, Tucson, 1992.

[88] N. Zingirian and M. Maresca, "On the efficiency of image and video processing programs on instruction level parallel processors," *Proceedings of the IEEE*, vol. 90, no. 7, pp. 1230-1243, July 2002.